AD-A265 254

FINAL TECHNICAL REPORT TO

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH

by

Jeffery L. Kennington
Department of Computer Science and Engineering
Southern Methodist University
Dallas, TX 75275-0122
(214) 768-3278

for

Optimization Algorithms for Integer Networks with Side Constraints for Application in Routing and Scheduling

February 22, 1993

AFOSR F49620-92-J-0032

SMU #5-25141

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

(2)

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)

2. REPORT DATE

3. REPORT TYPE AND DATES COVERED
FINAL/01 JAN 92 TO 31 DEC 92

4. TITLE AND SUBTITLE
OPTIMIZATION ALGORITHMS FOR INTEGER NETWORKS WITH SIDE CONSTRAINTS FOR APPLICATION IN ROUTING AND SCHEDULING

5. FUNDING NUMBERS
2304/DS
61102F

6. AUTHOR(S)
JEFFERY L. KENNINGTON

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
SOUTHERN METHODIST UNIVERSITY
RESEARCH ADMINISTRATION
P.O. BOX 8473
DALLAS TX 75275

8. PERFORMING ORGANIZATION REPORT NUMBER
AFOSR-TR- 93 0295

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR/NM
110 DUNCAN AVE, SUTE B115
BOLLING AFB DC 20332-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
F49620-92-J-0032

DTIC
ELECTE
MAY 14 1993
S B D

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE: DISTRIBUTION IS UNLIMITED

12b. DISTRIBUTION CODE

13. ABSTRACT (Maximum 200 words)

This document presents a new serial and parallei algorithms for the on-to-one shortest problem. This is the current best algorithms for this problem and we believe that our software implementation is the world's fastest code. Other algorithms for various network models, including the pure network problem, the generalized problem, the multicommodity network problem with a piecewise linear convex cost function are also presented.

93-10664

93 5 12 150

14. SUBJECT TERMS

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL (UNLIMITED)

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

2

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for [illegible] gathering and maintaining the data needed, and completing and reviewing the collection of information. [illegible] collection of information, including suggestions for reducing this burden, to Washington [illegible] Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project [illegible]

1. AGENCY USE ONLY *(Leave blank)*

2. REPORT DATE

3. REPORT TYPE AND DATES COVERED
FINAL/01 JAN 92 TO 31 DEC 92

4. TITLE AND SUBTITLE
OPTIMIZATION ALGORITHMS FOR INTEGER NETWORKS WITH SIDE CONSTRAINTS FOR APPLICATION IN ROUTING AND SCHEDULING

5. FUNDING NUMBERS
2304/DS
61102F

6. AUTHOR(S)
JEFFERY L. KENNINGTON

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
SOUTHERN METHODIST UNIVERSITY
RESEARCH ADMINISTRATION
P.O. BOX 8473
DALLAS TX 75275

8. PERFORMING ORGANIZATION REPORT NUMBER
AFOSR-TR- 93 0295

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR/NM
110 DUNCAN AVE, SUTE B115
BOLLING AFB DC 20332-0001

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
F49620-92-J-0032

DTIC
ELECTE
MAY 14 1993
S B D

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
APPROVED FOR PUBLIC RELEASE: DISTRIBUTION IS UNLIMITED

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This document presents a new serial and parallel algorithms for the on-to-one shortest problem. This is the current best algorithms for this problem and we believe that our software implementation is the world's fastest code. Other algorithms for various network models, including the pure network problem, the generalized problem, the multicommodity network problem with a piecewise linear convex cost function are also presented.

93-10664

93 5 12 1 0

14. SUBJECT TERMS

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT
UNCLASSIFIED

18. SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

19. SECURITY CLASSIFICATION OF ABSTRACT
UNCLASSIFIED

20. LIMITATION OF ABSTRACT
UL (UNLIMITED)

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
unclassified

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE

3. DISTRIBUTION/AVAILABILITY OF REPORT
unrestricted

4. PERFORMING ORGANIZATION REPORT NUMBER(S)

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
SMU

6b. OFFICE SYMBOL *(If applicable)*
CSE

7a. NAME OF MONITORING ORGANIZATION

6c. ADDRESS *(City, State and ZIP Code)*
Dallas, TX 75275-0122

7b. ADDRESS *(City, State and ZIP Code)*

8a. NAME OF FUNDING/SPONSORING ORGANIZATION
AFOSR

8b. OFFICE SYMBOL *(If applicable)*
NM

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER

8c. ADDRESS *(City, State and ZIP Code)*
110 Duncan Avenue, Suite 100
Bolling AFB, DC 20332-0001

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT NO

11. TITLE *(Include Security Classification)*

12. PERSONAL AUTHOR(S) Jeffery L. Kennington

13a. TYPE OF REPORT
Final

13b. TIME COVERED
FROM 1 Jan 92 TO 31 Dec92

14. DATE OF REPORT *(Yr., Mo., Day)*
22 Feb 1993

15. PAGE COUNT

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB. GR.

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

This document presents new serial and parallel algorithms for the one-to-one shortest path problem. This is the current best algorithm for this problem and we believe that our software implementation is the world's fastest code. Other algorithms for various network models, including the pure network problem, the generalized network problem, the multicommodity network problem, and the minimum cost network problem with a piecewise linear convex cost function are also presented.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT. □ DTIC USERS □

21. ABSTRACT SECURITY CLASSIFICATION
Unclassified

22a. NAME OF RESPONSIBLE INDIVIDUAL
Jeffery L. Kennington

22b. TELEPHONE NUMBER *(Include Area Code)*
214/768-3278

22c. OFFICE SYMBOL
CSE

EDITION OF 1 JAN 73 IS OBSOLETE.

SECURITY CLASSIFICATION OF THIS PAGE

Table of Contents

DTIC QUALITY INSPECTED 5

Accession For

NTIS CRA&I ☑

DTIC TAB □

Unannounced □

Justification

By

Distribution/

Availability Codes

Dist	Avail and/or Special
A-1	

I. Statement of Work

Many United States Air Force routing and scheduling problems can be analyzed and modelled mathematically as some type of network flow problem. As budget pressures become more intense, there will be even more emphasis on optimizing the use of the assets available to the Air Force. Optimization algorithms and state-of-the-art software will be important tools that can be used by Air Force planners and decision-makers during the foreseeable future. Models and techniques which exploit the underlying network structure of these problems will also gain added importance. The research reported in this document is directed toward the discovery of new and improved algorithms and the development of computational software to assist in the area of routing and scheduling problems.

II. Technical Reports

Title

Network Flows
(September 1992)

Authors

Richard V. Helgason
Jeffery L. Kennington

Executive Sumary

The objective of this report is to clearly present the techniques used in a computationally efficient implementation of the primal simplex algorithm specialized for various network models. Numerous examples are included to help illustrate the ideas and techniques. It is shown mathematically and illustrated by example that every basis for a linear network problem is triangularizable and corresponds to a spanning tree. This result is used in the development of a specialization of the primal simplex algorithm for the linear network problem which is at least two orders of magnitude faster then general purpose software. It is shown mathematically and illustrated by examples that every basis for a generalized network problem has a block diagonal structure with each block corresponding to either a rooted tree or a one-tree. While this structure is not as beneficial as that of the linear network problem, it can still be exploited in performing the simplex operations. Other similar results for the multicommodity network flow problem are also presented.

Publication Status

This manuscript has been submitted for publication as a chapter in a book and is currently under review.

Title

The One-to-One Shortest-Path Problem: An Empirical Analysis with the Two-Tree Dijkstra Algorithm
(Revised December 1992)

Authors

Richard V. Helgason
Jeffery L. Kennington
B. Douglas Stewart

Executive Summary

The problem of finding the shortest path between a designated pair of nodes in a graph is a fundamental problem in operations research, computer science, and scheduling theory. Good algorithms for this problem can be used as a building block for other algorithms to solve the assignment problem, the semi-assignment problem, the multicommodity network flow problem, and integer networks with side constraints. The classical Dijkstra algorithm begins at one of the designated nodes and fans out from this node until the other designated node becomes a member of the labeled set. This is easily accomplished by a computer implementation that builds a tree rooted at one of the designated nodes. In our investigation we empirically demonstrate that a better algorithm (in terms of computational time) is obtained by a procedure that begins at both designated nodes and fans out in both directions, either simultaneously as in our parallel implementation or alternately as in our sequential implementation. This new algorithm terminates when any node appears in the labeled set for both trees. An interesting feature of this procedure is that the node which first appears in both trees may not be present in the shortest-path, but enough information has been developed to find the shortest-path. The speed improvement of this new algorithm results from the empirical observation that many fewer nodes need to be scanned under the new scheme. The new scheme is more complicated to implement, but results in a substantially faster software package.

Publication Status

This paper has been accepted for publication in Computational Optimization and Application.

Title

A Direct Simplex Algorithm for the Network Flow Problem with Piecewise Linear Costs
(January 1993)

Authors

Rajluxmi V. Murthy
Richard V. Helgason

Executive Summary

This paper presents a specialization of the simplex algorithm for a network problem having a cost function which is piecewise linear and convex. The basis characterization is the same for this problem but an improvement can be made in the simplex pricing operation. The specialization results in at least a 50% speed improvement over existing methods for this problem.

Publication Status

This paper has been submitted for publication and is under review.

APPENDIX A

NETWORK FLOWS

Richard V. Helgason
Jeffery L. Kennington

Department of Computer Science and Engineering
Southern Methodist University
Dallas, Texas 75275

15 September 1992

1 Introduction

Our aim is to (1) summarize the ideas fundamental to efficient implementation of the primal network simplex algorithm and its extensions, and (2) indicate how these algorithms may be effectively used within other optimization algorithms.

1.1 Set Notation

For the most part we adopt standard set notation conventions. Sets will usually be denoted by upper case Roman letters such as X. The empty set will be denoted by Φ. For a finite set X we let $\#X$ denote the number of elements in X. We let $\imath n = \{1, \ldots, n\}$ and $\jmath n = \{0, ..., n\}$. Given set X, we define the *equality relation on* X to be $X \doteq X \equiv \{(x,x) : x \in X\}$. We will also use multisets in which a repetition factor is allowed for set elements. For a finite multiset Y then, $\#Y$ will also incorporate multiplicities.

1.2 Matrix and Vector Notation

Matrices will usually be denoted by upper case Roman letters such as A. Row vectors will usually be denoted by lower case Greek letters such as π. Column vectors will usually be denoted by lower case Roman letters such as x. The element in the ith row and jth column of matrix A will be denoted by A_{ij}. The row (column) vector whose entries are from the ith row (jth column) of A will be denoted by $A_{i.}$ $(A_{.j})$. The ith element of a vector such as x will be denoted by x_i. We will allow extensive subscripting and superscripting of matrices and vectors for identification purposes. Inasmuch as this may interfere with the subscripting convention for element identification above, we also adopt the functional notation $(\cdot)_i$ and $(\cdot)_{ij}$ for vector and matrix element identification, respectively, so that $(X_{ij})_{pq}$ is the pqth element of matrix X_{ij}. We will use e_i (e^i) for the column (row) vector whose ith element is a 1 and whose other elements are all zeros. We will use e_{ij} to denote the column vector whose ith element is a 1, whose jth element is a -1, and whose other elements are all zeros, so that e_{ij} is $e_i - e_j$. We will use $\hat{0}$ and $\hat{1}$ as row or column vectors with orientation and dimension given by context, having as uniform elements 0 or 1, respectively. We abuse notation by allowing $\imath n = \{1, \ldots, n\}$ to also be used as a row vector. The *diagonal* of matrix A is the set of elements $\{A_{ii}\}$. The matrix A is said to be *upper* (*lower*) *triangular* if $A_{ij} = 0$ when $j > i$ ($i > j$) and, more simply, *triangular* in either case. The matrix A is said to be *diagonal* if it is both upper and lower triangular. A triangular matrix will be nonsingular when its diagonal elements are all nonzero. The matrix A is said to be *triangularizable* if it can be brought to nonsingular triangular form by a sequence of row and column interchanges.

1.3 Graph Notation

We define a set of *nodes* or *vertices* V to be any set of consecutive integers which we typically take to be $\imath n$ or $\jmath n$. Given a set of nodes V, we define an *arc* or *edge* for V to be any ordered pair (i, j) with $i \in V$, $j \in V$, and $i \neq j$. The arc (i, j) is said to be *incident on* (touch) both i and j, to *connect* i and j (or j and i), and to be *directed from i to j*. Formally, a *network* or *directed graph* is defined to be $G = < V, E >$ where V is a set of nodes and E is a set of arcs for V. Apparently then $E \subseteq (V \times V) \backslash (V \doteq V)$. When $V = \Phi$ then also $E = \Phi$ and in this case G is called the *trivial* graph. We shall also allow E to be a multiset when it is desirable to have more than one arc connect two nodes. In this case one could more properly refer to G as a *multigraph*. For $\# E = m$, we will find it convenient to label the arcs with elements from $\imath m$.

1.4 Visual Representation

The nodes of a network may be viewed as locations or terminals where a given commodity can be moved from, to, or through and the arcs of a network may be viewed as unidirectional means of commodity transport connecting or serving those nodes. Hence arcs may represent streets and highways in an urban transportation network, pipes in a water distribution network, or telephone lines in a communication network. The structure of the network can be displayed by means of a labeled drawing in which nodes are represented by circles or boxes and arcs are represented by line segments incident on two nodes. Each line segment will have an arrowhead placed somewhere on it to indicate the direction of the associated commodity transport. Typically the arrowhead will be incident on the node to which the commodity is being transported. An example network illustration is given in Figure 1.

Figure about here

Figure 1: Example network

1.5 Node-Arc Matrix Representation

The structure of a network may also be described using a *node-arc incidence matrix* A given by

$$A_{ik} = \begin{cases} +1 & \text{if arc } k \text{ is directed away from node } i, \\ -1 & \text{if arc } k \text{ is directed toward node } i, \\ 0 & \text{otherwise.} \end{cases}$$

Apparently then $A_{.k} = e_{ij}$ for some i and j, and we shall allow ourselves to abuse notation by saying that in this case the kth arc *is* e_{ij}. An example node-arc incidence matrix corresponding to Figure 1 is given below.

$$\begin{array}{cc} & \underline{\text{arcs}} \\ & \begin{array}{ccccccc} 1 & 2 & 3 & 4 & 5 & 6 & 7 \end{array} \\ \underline{\text{nodes}} \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \left(\begin{array}{ccccccc} 1 & 1 & 0 & 0 & 0 & 0 & -1 \\ -1 & -1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & -1 & -1 & 1 & 1 \end{array}\right) \end{array}$$

1.6 Subgraphs

A graph $G' =< V', E' >$ is said to be a *subgraph* of $G =< V, E >$ if $V' \subseteq V$ and $E' \subseteq E$. Note that G' is required to be a graph itself, so that V' and E' cannot simply be arbitrary subsets of V and E, respectively. Further, G' is said to *span* G or G' is said to be a *spanning subgraph* for G when $E' = E$. Given a node subset $V' \subseteq V$, we define the *subgraph generated by* V' to be $G(V') \equiv \{(i,j) \in G : i \in V' \text{ and } j \in V'\}$. Example subgraphs corresponding to Figure 1 are given in Figure 2.

Figure about here

Figure 2: Example network subgraphs

1.7 Paths and Cycles

Given a graph $G =< V, E >$, a finite odd length sequence

$$P = \{v_1, e_{i_1 j_1}, v_2, e_{i_2 j_2}, \ldots, v_q, e_{i_q j_q}, v_{q+1}\}$$

whose odd elements are nodes of V and whose even elements are arcs of E is defined to be a *walk* in G of *length* $q \geq 0$ in G if: (1) P has at least one node, and (2) for $0 < r \leq q$, arc $e_{i_rj_r}$ connects v_r and v_{r+1}. Apparently then from (2), $e_{i_rj_r}$ could be either (v_r, v_{r+1}) or (v_{r+1}, v_r). The sequence formed by reversing the elements of P is also a walk and will be denoted by $rev(P)$. If we envision moving from v_1 *to* v_{q+1}, utilizing the sequential elements of P in order, we can assign an (implied) orientation to the arcs in the walk by defining the *orientation function*

$$O(e_{i_rj_r}) = \begin{cases} +1 & \text{if } e_{i_rj_r} = (v_r, v_{r+1}), \\ -1 & \text{if } e_{i_rj_r} = (v_{r+1}, v_r). \end{cases}$$

If the sequence of nodes $\{v_1, \ldots, v_{q+1}\}$ from P is composed of distinct nodes, the walk P is said to be a *(simple) path* which *links* v_1 *to* v_{q+1}. It follows that the arcs of a path are distinct. Apparently then $rev(P)$ is also a path which links v_{q+1} *to* v_1. It also follows that any walk P of length 0 is a path which links v_1 to itself. If the walk P (1) is of length at least two, (2) $\{v_1, e_{i_1j_1}, \ldots, v_q\}$ is a path, (3) $\{v_2, e_{i_2j_2}, \ldots, v_{q+1}\}$ is a path, and (4) $v_1 = v_{q+1}$, the walk P is said to be a *cycle*. Example walks in the graph of Figure 1 are given in Figure 3.

Figure about here

Figure 3: Walks in the example network

Given a cycle P it is possible to form other cycles using the sequential elements of P in wrap-around order, i.e., starting at v_m we can define a cycle

$$P' = \{v_m, e_{i_mj_m}, v_{m+1}, \ldots, v_q, e_{i_qj_q}, v_1, e_{i_1j_1}, v_2, \ldots, v_m\}$$

which retains the essential arc and node orders and arc orientations of P when we envision moving from v_m *to* v_m on P'. Thus we will consider cycles such as P and P' to be the same cycle and also refer to any of this set of equivalent representations as *a cycle on nodes* $\{v_1, \ldots, v_q\}$. The arcs of a cycle are generally distinct, except for two special cases which can arise when considering cycles of length two. Cycles of the form

$$\{v_1, (v_1, v_2), v_2, (v_1, v_2), v_1\}$$

and

$$\{v_1, (v_2, v_1), v_2, (v_2, v_1), v_1\}$$

do have arcs which are not distinct and will be called *inadmissable* cycles. All other cycles (which have distinct arcs) will be called *admissable*. Apparently then if P is an admissable cycle on nodes $\{v_1, \ldots, v_q\}$ then $rev(P)$ is a distinct cycle on nodes $\{v_1, \ldots, v_q\}$. A graph G in which no admissable cycle can be formed is said to be *acyclic* and is also said to *contain no cycles*.

1.8 Connectedness and Components

A graph $G =< V, E >$ is said to be connected if any two vertices u and v can be linked by a path in G. The maximal connected subgraphs of G serve to partition G and are called *components* of G. If $G' =< \{v\}, \Phi >$ is a component of G, v is said to be an *isolated* node of G.

1.9 Trees

A nontrivial connected acyclic graph is called a *tree*. A graph which consists of an isolated node only will be called a *trivial tree*. A graph whose components are all trees is called a *forest*. An *endnode* of a tree is a node which has only one arc of the tree incident on it. A *leaf* of a tree is an arc of the tree incident on an endnode.
A tree $G =< V, E >$ has several important properties:

(1) E has one less arc than V has nodes, i.e. $\#E = \#V - 1$,

(2) if an endnode and a leaf incident on it are removed from G, the resulting subgraph is also a tree,

(3) if G has an arc (i, j) incident on two endnodes, then $V = \{i, j\}$ and $E = \{(i, j)\}$ or $E = \{(j, i)\}$.

(4) if $\#E = 1$, G has exactly one leaf,

(5) if $\#E > 1$, G has at least two leaves,

(6) for every distinct pair of nodes $u, v \in E$, there is a unique path in G linking u to v.

An example tree is given in Figure 4.

Figure about here

Figure 4: Example tree

A *root* is a node of a tree which we wish to distinguish from the others, usually for some algorithmic purpose. Occasionally this may be made explicit by drawing a tree oriented with the root at the top of the diagram. Alternatively, this may be made explicit in a diagram by drawing a short line segment (with or without an arrowhead) incident on only the root node. An example rooted tree is given in Figure 5.

Figure about here

Figure 5: Example rooted tree

1.10 Tree Solution Algebra

Consider the solution of the system

$$Ax = b, \qquad (1)$$

where A is the $n \times (n-1)$ node-arc incidence matrix for a tree $T =< V, E >$ with n nodes and $n-1$ arcs and b is a given n-vector. A procedure which can be used to reduce the system to a permuted diagonal form is given below:

```
                    procedure DIAGONAL REDUCTION

inputs:   $T =< V, E >$   -  nontrivial tree with $n$ nodes
                $\rho$         -  root node for T
                $A$         -  node-arc incidence matrix for T
                $b$         -  $n$-vector

output:      $\bar{A}x = \bar{b}$    -  diagonalized system equivalent to $Ax = b$
begin
  [Initialize]
  $\bar{V} \Leftarrow V, \bar{E} \Leftarrow E, \bar{A} \Leftarrow A, \bar{b} \Leftarrow b$ ;
  [Iterate until the tree becomes trivial]
  while $\bar{E} \neq \Phi$ do
    [Pick a leaf]
    select an endnode $p$ not the root $\rho$ in the tree $\bar{T} =< \bar{V}, \bar{E} >$.
    let $r$ be the other node of the leaf incident on $p$.
    let $(i,j)$ be the leaf incident on $p$ and $r$.
    let $c$ be the column of $\bar{A}$ corresponding to the leaf $(i,j)$ .
    [Pivot the system on the selected endnode]
    $\bar{b}_r \Leftarrow \bar{b}_r + \bar{b}_p, \bar{A}_{r.} \Leftarrow \bar{A}_{r.} + \bar{A}_{p.}$;
    $\bar{b}_p \Leftarrow \bar{A}_{pc}(\bar{b}_p), \bar{A}_{p.} \Leftarrow \bar{A}_{pc}(\bar{A}_{p.})$;
    [Update the tree by removing the leaf]
    $V \Leftarrow V \setminus \{p\}, E \Leftarrow E \setminus \{(i,j)\}$;
  endwhile
end
```

Note that at each pivot step in the above procedure, a partial sum of components of b is produced in the component $\bar{b}_r$, and in the last pivot, $\bar{b}_\rho = \sum_{k=1}^{n} b_k$ is produced. Also, after each pivot and tree update, the subset of the rows and columns of $\bar{A}$ corresponding to the nodes of $\bar{V}$ and the arcs of $\bar{E}$, respectively, is the node-arc incidence matrix of the updated tree $\bar{T}$.

In a node-arc incidence matrix A for a tree, let c be the column corresponding to a leaf with an endnode p not the root node ρ and let r be the other leaf node. Row p contains only one nonzero in column c, and row r contains the only other nonzero in column c. Thus when a pivot is made at the matrix position A_{pc}, A_{rc} will be zeroed and A_{pc} will become 1 if it was -1, but no other entries in A will be altered.

Now $\bar{A}$ initially has $2n-2$ nonzeros and $n-1$ pivots are made overall, so that the final pivot produces a $\bar{A}$ with $n-1$ nonzeros, all in the pivot rows. Thus row ρ of $\bar{A}$, the only row in which no pivot occurred, must contain all zeros. It follows that the system (1) has a solution if and only if $\bar{b}_\rho = 0$, hence no solution is possible unless $\sum_{k=1}^{n} b_k = 0$. Furthermore, since $n-1$ pivots have been made, the matrix A has rank $n-1$ so that when $\sum_{k=1}^{n} b_k = 0$, the solution produced by the algorithm is the unique solution to system (1).

To illustrate the use of the algorithm consider the tree in Figure 4. The original system corresponding to (1) is

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

Selecting node 4 as the root and using the sequence of selected endnodes $3, 2, 1$ produces the following sequence of systems.

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & -1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 + b_3 \\ -b_3 \\ b_4 \end{bmatrix}$$

$$\begin{bmatrix} -1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 + b_3 \\ -b_3 \\ b_2 + b_3 + b_4 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} -b_1 \\ b_2 + b_3 \\ -b_3 \\ b_1 + b_2 + b_3 + b_4 \end{bmatrix}$$

Now let us consider adjoining an additional column e_k, where $1 \leq k \leq n$, to the right of A and lengthening x to accomodate the additional column. The expanded system is then

$$[A|e_k]x = b, \tag{2}$$

Suppose that we agree to choose k as the root node ($\rho = k$) and apply the above procedure to the expanded matrix and original tree, i.e. in the initialization step we set $\bar{A} \Leftarrow [A|e_\rho]$ instead of $\bar{A} \Leftarrow A$.

The same vector $\bar{b}$ and matrix $\bar{A}$ is produced in the first $n-1$ (original) columns and $\bar{A}_{.n} = e_k = e_\rho$. The system (2) is also permuted diagonal but is now of rank n and in its solution has $e_k = \sum_{k=1}^{n} b_k$. Since $[A|e_k]$ is square, the solution produced by the procedure must be unique. Furthermore, the solution to (1) produced by the above procedure when $\sum_{k=1}^{n} b_k = 0$ must have the same first $n-1$ variable values as those produced for the enlarged system (2).

In the above example, the original system corresponding to (2) with root node 4 is

$$\begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & -1 & 0 & 0 \\ 1 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

And after the same sequence of pivots, the final equivalent system produced is

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} -b_1 \\ b_2 + b_3 \\ -b_3 \\ b_1 + b_2 + b_3 + b_4 \end{bmatrix}$$

We remark that this usage of an extra e_k column in conjunction with the solution of system (1) provides strong impetus to extend the node-arc matrix representation to include a representation of a root k by a column e_k, when the underlying graph is a tree. We will find it useful to do so even when the underlying graph is a tree with additional arcs.

2 Primal Simplex Algorithm

All the network models presented in this Chapter may be viewed as special cases of the linear program and many of the algorithms to be presented are specializations of the primal simplex algorithm. Familiarity with the simplex method is assumed and this section simply presents a concise statement of the primal simplex method which will be specialized for several of the network models.

Let A be an $m x n$ matrix with full row rank, c and u be n-component vectors, and b be an m-component vector. Let $X = \{x : Ax = b, \hat{0} \leq x \leq u\}$ and assume that $X \neq \Phi$. The *linear program* is to find an n-component vector $\bar{x}$ such that $c\bar{x} = \min\{cx : x \in X\}$. We adopt the convention that the symbol x refers to the vector of unknown decision variables and the symbol $\bar{x}$ refers to some known point in X.

Since A has full row rank, by a sequence of column interchanges it may be displayed in the partitioned form $[B\,|\,N]$, where B is nonsingular. With a corresponding partitioning of both x and u, we may write

$$X = \{(x^B \,|\, x^N) : Bx^B + Nx^N = b\,, \hat{0} \leq x^B \leq u^B, \hat{0} \leq x^N \leq u^N\}$$

A point $(\bar{x}^B \,|\, \bar{x}^N) \in X$ is called a *basic feasible solution* if $\bar{x}_j^N \in \{0, u_j^N\}$ for all $j \in \imath(n-m)$. The variables in x^B (x^N) are called the *basic* (*nonbasic*) variables. It is well known from linear programming theory that for every linear program with $X \neq \Phi$, there exists at least one basic feasible solution which is optimal.

We say that two basic feasible solutions are *adjacent* if x^{B_1} and x^{B_2} differ by exactly one variable, so that B_1 may differ from B_2 in exactly one column. The primal simplex algorithm is an iterative procedure which begins with some basic feasible solution and moves through a sequence of adjacent basic feasible solutions until a stopping criterion is met which guarantees that the final basic feasible solution is optimal.

A mathematical description of the primal simplex algorithm follows:

```
                    procedure PRIMAL SIMPLEX

inputs:       $A$ - $m \times n$ constraint matrix
              $c$ - $m$-component vector of unit costs
              $u$ - $n$-component vector of upper bounds
              $b$ - $m$-component right-hand-side vector

output:       $(\bar{x}^B \,|\, \bar{x}^N)$, an optimal basic feasible solution for min$\{cx : x \in X\}$

assumptions:  1. $rank(A) = m$
              2. $X = \{x : Ax = b\,, \hat{0} \leq x \leq u\} \neq \Phi$

begin
  [Initialization]
  optimal $\Leftarrow$ 'no';
  let $(\bar{x}^B \,|\, \bar{x}^N) \in X$ be any basic feasible solution with $A$, $c$, and $u$ parti-
     tioned in corresponding fashion as $[B \,|\, N]$, $(c^B \,|\, c^N)$, and $(u^B \,|\, u^N)$;
  while optimal = 'no' do
    [Dual Calculation]

                    $\pi \Leftarrow c^B B^{-1}$

    [Pricing]

              $L \Leftarrow \{j : \pi N_{\cdot j} - c_j^N > 0 \text{ and } \bar{x}_j^N = 0\};$
              $M \Leftarrow \{j : \pi N_{\cdot j} - c_j^N < 0 \text{ and } \bar{x}_j^N = u_j^N\};$
```

```
if $L \cup M = \Phi$
  then
    optimal $\Leftarrow$ 'yes'
  else
    select $k \in L \cup M$ ;
    [Column Update]

                    $y \Leftarrow B^{-1} N_{\cdot k};$

    [Ratio Test]

              $R \Leftarrow \{i : y_i > 0\}$ and $S \Leftarrow \{i : y_i < 0\};$

    if $k \in L$
      then

      $\Delta \Leftarrow \min\{u_k^N, \min_{i \in R} \left(\frac{x_i^B}{y_i}\right), \min_{i \in S} \left(\frac{u_i^B - x_i^B}{-y_i}\right)\};$
      $T \Leftarrow \{i \in R : \left(\frac{x_i^B}{y_i}\right) = \Delta\} \cup \{i \in S : \left(\frac{u_i^B - x_i^B}{-y_i}\right) = \Delta\};$

      else

      $\Delta \Leftarrow \min\{u_k^N, \min_{i \in R} \left(\frac{u_i^B - x_i^B}{y_i}\right), \min_{i \in S} \left(\frac{x_i^B}{-y_i}\right)\};$
      $T \Leftarrow \{i \in R : \left(\frac{u_i^B - x_i^B}{y_i}\right) = \Delta\} \cup \{i \in S : \left(\frac{x_i^B}{-y_i}\right) = \Delta\};$
    endif
    [Value Update]

    if $k \in L$
      then

              $\bar{x}_k^N \Leftarrow \Delta$ and $\bar{x}^B \Leftarrow \bar{x}^B - \Delta y;$

      else

            $\bar{x}_k^N \Leftarrow u_k^N - \Delta$ and $\bar{x}^B \Leftarrow \bar{x}^B + \Delta y;$
    endif
    if $\Delta \neq u_k^N$
      then
        [Basis Exchange Update]

        select $j \in T$ ;
        Interchange $x_j^B$ with $x_k^N$ to form a new
          partitioning $[B \mid N]$ ;
    endif
    endif
  endwhile
end
```

The only sticky issue concerning the above algorithm is that it may be possible to move through a sequence of basis interchanges, all of which correspond to the same actual point, only changing the representations to correspond to the varying bases. That is, there may exist a sequence of basis interchanges all having $\Delta = 0$ which could result in the above procedure being stuck in a nonterminating loop. This phenomenon is known as *cycling*. Much discussion of cycling can be found in the literature along with anticycling rules[1] which have been developed to insure convergence.

3 Linear Network Models

Let $< V, E >$ be a network through which some commodity will be flowing. Associated with each node $v \in V$ we define a requirement r_v A node having a supply of the commodity is assigned a positive requirement equal to the supply. Conversely, negative requirements correspond to demands for the commodity at the specified nodes. The requirements for all ***transshipment nodes*** are zero.

Suppose that for the example network illustrated in Figure 1, that nodes 1 and 3 are supply nodes with supply of 10 and 5, resp tively; and that node 4 is a demand point with a demand of 15. This netwoı and the corresponding requirements are illustrated in Figure 6.

Figure about here

Figure 6: Example network with requirements

For linear network problems, one is seeking a set of flows on the arcs which satisfy the supply and demand restrictions at each node. For the example network, the total flow into node 2 plus the five supply units which originate at node 2 must equal the total flow departing node 2. We say that a feasible flow satisfies ***flow conservation*** which implies that it is an element of $\{x : Ax = r\}$, where A is the corresponding node-arc incidence matrix for $< V, E >$.

[1] We have never observed cycling in any of our software implementations of this algorithm and have not incorporated any of the anticycling rules in our software.

For the example network, the flow conservation equations are

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & -1 \\ -1 & -1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & -1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} 10 \\ 5 \\ 0 \\ -15 \end{bmatrix}$$

Associated with each of the arcs in E, we define a vector of unit flow costs c and a vector of flow capacities or bounds u. Thus, the cost for each unit of flow in arc k is given by c_k and the flow on arc k is restricted to the interval $[0, u_k]$. Mathematically, the linear network model on $< V, E >$ with node-arc incidence matrix A is given by

$$\min_x \{cx : Ax = r, 0 \leq x \leq u\}$$

A sample model for the network illustrated in Figure 6 is

minimize $x_1 + 3x_2 + 5x_3 - 7x_4 + 7x_5 - x_6 + 9x_7$
subject to

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & -1 \\ -1 & -1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & -1 & -1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} = \begin{bmatrix} 10 \\ 5 \\ 0 \\ -15 \end{bmatrix}$$

$$\begin{array}{ccccc} 0 & \leq & x_1 & \leq & 6 \\ 0 & \leq & x_2 & \leq & 8 \\ 0 & \leq & x_3 & \leq & 10 \\ 0 & \leq & x_4 & \leq & 10 \\ 0 & \leq & x_5 & \leq & 8 \\ 0 & \leq & x_6 & \leq & 8 \\ 0 & \leq & x_7 & \leq & 8 \end{array}$$

3.1 Basis Characterization

Recall that one of the assumptions for the primal simplex algorithm is that the constraint matrix has full row rank, i.e. the $m \times n$ constraint matrix A has $rank(A) = m$. But it is clear that if A is a node-arc incidence matrix, $rank(A) < m$ since for a connected graph, the rank of a node-arc incidence matrix is one less than the number of nodes (also $\hat{1}A = \hat{0}$). Traditionally, a *root arc* is appended to the problem so that the constraint matrix will have full row rank. Let

$$X^E = \{(x|a) : [A|e_\rho] \begin{bmatrix} x \\ - \\ a \end{bmatrix} = r, 0 \leq x \leq u\}$$

The revised model is then simply

$$\min_x \{cx : (x|a) \in X^E\},$$

where the enlarged constraint matrix $[A|e_\rho]$ has full row rank, so that the primal simplex algorithm can be applied directly to this model. In Section 1.10 it was shown that the root arc will carry no flow ($a = 0$) when $\sum_i r_i = 0$ and we are only solving the linear system. This will also be true when any nonbasic arcs are set to upper bound, since setting a nonbasic flow $x_k = u_k$, where x_k is the flow on arc (i,j) is equivalent to adding u_k to r_j and subtracting u_k from r_i, thus preserving the condition $\sum_i r_i = 0$.

Let B be a basis for $[A|e_\rho]$ so that the entire matrix is partitioned as $[B|N]$. B may be further partitioned into $[S|e_\rho]$. Recall that the arcs corresponding to S must form a spanning tree from $< V, E >$. We may write the corresponding basic solution as $(x^S|a|x^N)$.

By row and column interchanges B may be displayed in lower triangular form. The trees corresponding to the bases

$$\begin{array}{cccc} e_{41} & e_{24} & e_{23} & e_2 \\ \left[\begin{array}{c} -1 \\ 0 \\ 0 \\ 1 \end{array}\right. & \begin{array}{c} 0 \\ 1 \\ 0 \\ -1 \end{array} & \begin{array}{c} 0 \\ 1 \\ -1 \\ 0 \end{array} & \left.\begin{array}{c} 0 \\ 1 \\ 0 \\ 0 \end{array}\right] \end{array} \text{ and } \begin{array}{cccc} e_{12} & e_{41} & e_{43} & e_3 \\ \left[\begin{array}{c} 1 \\ -1 \\ 0 \\ 0 \end{array}\right. & \begin{array}{c} -1 \\ 0 \\ 0 \\ 1 \end{array} & \begin{array}{c} 0 \\ 0 \\ -1 \\ 1 \end{array} & \left.\begin{array}{c} 0 \\ 0 \\ 1 \\ 0 \end{array}\right] \end{array}$$

are illustrated in Figure 7.

Figure about here

Figure 7: Basis trees for sample network

The trees can be used to determine the row and column interchanges required to display the bases in lower triangular form. The root node and root arc are always placed in the last row and column. The first $m - 1$ rows and columns are determined recursively by a process in which we first select an endnode with its corresponding leaf arc and then remove them from the tree after appropriately reordering the matrix. Note that this ordering may not be unique since multiple endnodes are always present.

A procedure which can be used to display a linear network basis in lower triangular form is given below:

procedure DISPLAY LOWER TRIANGULAR

input: $[S|e_\rho]$ - $m \times m$ basis for a linear network problem

outputs: $norder[i]$ - the node in row position i
$arcorder[j]$ - the arc in column position j

assumption: S corresponds to a spanning tree

```
begin
 [Initialization]
 let $< V, T >$ be a network corresponding to $S$
 $nodeorder[m] \Leftarrow \rho$, $arcorder[m] \Leftarrow e_\rho$;
 $i \Leftarrow 1$;
 while $i < m$ do
   [Tree Reduction]
   let $v \in V$ be an endnode of $< V, T >$;
   let $e_{jk}$ be the leaf arc incident to $v$;
   $nodeorder[i] \Leftarrow v$, $arcorder[i] \Leftarrow e_{jk}$, $V \Leftarrow V \backslash \{v\}$, $T \Leftarrow T \backslash \{e_{jk}\}$, $i \Leftarrow i+1$;
 endwhile
end
```

An app' cation of procedure DISPLAY LOWER TRIANGULAR to the tree in Figure 7a with certain choices in the tree reduction yields *nodeorder* = $[3, 1, 4, 2]$ and *arcorder* = $[e_{23}, e_{41}, e_{24}, e_2]$. The corresponding matrix is:

$$\begin{array}{c|cccc|} (nodes) & e_{23} & e_{41} & e_{24} & e_2 \\ 3 & -1 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 \\ 4 & 0 & 1 & -1 & 0 \\ 2 & 1 & 0 & 1 & 1 \end{array}$$

Another application of the procedure with different choices made in the tree reduction would yield *nodeorder* = $[1, 3, 4, 2]$ and *arcorder* = $[e_{41}, e_{23}, e_{24}, e_2]$, with corresponding matrix is:

$$\begin{array}{c|cccc|} (nodes) & e_{41} & e_{23} & e_{24} & e_2 \\ 1 & -1 & 0 & 0 & 0 \\ 3 & 0 & -1 & 0 & 0 \\ 4 & 1 & 0 & -1 & 0 \\ 2 & 0 & 1 & 1 & 1 \end{array}$$

3.2 Dual Calculation

Based on the results in Section 3.2, it is known that every basis for a linear network problem takes the form $[S|e_\rho]$, where the columns of S correspond to a spanning tree $< V, T >$ and S is triangularizable. Therefore the dual calculation

$$\pi[S|e_\rho] = [c^S|0]$$

can be specialized to exploit the underlying network structure. Since S consists of columns from a node-arc incidence matrix, every column of S is a vector e_{ij} for some i and j. Hence $S = [e_{i_1j_1}, e_{i_2j_2}, \ldots, e_{i_{m-1}j_{m-1}}]$ and the dual calculation reduces to the system

$$\left\{ \begin{array}{ccc} \pi_{i_1} - \pi_{j_1} & = & c^S_1 \\ \pi_{i_2} - \pi_{j_2} & = & c^S_2 \\ \vdots & \vdots & \vdots \\ \pi_{i_{m-1}} - \pi_{j_{m-1}} & = & c^S_{m-1} \\ \pi_\rho & = & 0 \end{array} \right\}$$

Since this system is upper triangular it can be solved by back substitution. Beginning with $\pi_\rho = 0$, the duals for all basic arcs incident on node ρ can be determined. Once these duals are known, all duals for basic arcs incident on those arcs can be determined. Continuing in this manner, eventually all duals will be determined. The following procedure formally states how this can be accomplished without actually triangularizing a matrix.

procedure DUAL CALCULATION

inputs:	$[S|e_\rho]$	-	$m \times m$ basis for a linear network problem
	c^S	-	$m-1$ vector of basic costs, where c^S_{ij} is the unit cost for basic arc e_{ij}
output:	π	-	$\pi = [c^S|0][S|e_\rho]^{-1}$

```
 begin
[Initialization]
let $< V, T >$ be the spanning tree corresponding to $S$;
for $i = 1, \ldots, m$
  $label[i]$ = 'no';
endfor
$\pi_\rho \Leftarrow 0$, $label[\rho] \Leftarrow$ 'yes', $k \Leftarrow 1$;
```

```
  while $k < m$ do
    let $e_{ij} \in T$ such that $label[i] \neq label[j]$;
    if $label[i] =$ 'yes'
      then
        $\pi_j = \pi_i - c^S_{ij}, label[j] =$ 'yes';
      else
        $\pi_i = \pi_j + c^S_{ij}, label[i] =$ 'yes';
    endif
    $k \Leftarrow k + 1$;
  endwhile
end
```

Consider the basis tree in Figure 8 with

$$[c^S_{12}, c^S_{34}, c^S_{41}, c^S_{51}, c^S_{74}, c^S_{76}] = [1, 2, 3, -1, -2, -3]$$

Since node 4 is the root node, $\pi_4 = 0$. The equations

$$\left\{ \begin{array}{rcr} \pi_3 - \pi_4 & = & 2 \\ \pi_4 - \pi_1 & = & 3 \\ \pi_7 - \pi_4 & = & -2 \end{array} \right\}$$

yield $\pi_3 = 2, \pi_1 = -3$, and $\pi_7 = -2$. Then the equations

$$\left\{ \begin{array}{rcr} \pi_5 - \pi_1 & = & -1 \\ \pi_1 - \pi_2 & = & 1 \\ \pi_7 - \pi_6 & = & -3 \end{array} \right\}$$

yield $\pi_5 = -4, \pi_2 = -4$, and $\pi_6 = 1$. Also note that the dual calculation only involves addition and subtraction. Hence, if the cost coefficients are all integer, then the dual variables will also be integral valued.

Figure about here

Figure 8: Dual variable calculation example

3.3 Column Update

Suppose e_{st} denotes the nonbasic arc which prices favorably and is selected for a potential flow change. There is no guarantee that the flow will actually change since the ratio test could yield $\Delta = 0$. The column update step of the primal simplex algorithm requires that $y = [S|e_p]^{-1}e_{st}$ be determined. Since the arcs of S correspond to a spanning tree and $[S|e_p]$ is lower triangular, the calculation of y can be simplified.

The updated column y is a vector which solves the lower triangular system $[S|e_\rho]y = e_{st}$ or, in component form,

$$S_{.1}y_1 + S_{.2}y_2 + \cdots + S_{.m-1}y_{m-1} + e_\rho y_m = e_{st}$$

That is, a set of scalars $y_1, \ldots, y_m$ are sought which when multiplied by the columns of S and the vector e_ρ and added together form the vector e_{st}. Let $< V, T >$ denote the tree corresponding to S and let

$$P = \{v_1, e_{i_1j_1}, v_2, e_{i_2j_2}, \ldots, v_q, e_{i_qj_q}, v_{q+1}\}$$

denote the simple path in $< V, T >$ linking s to t. Such a path is illustrated in Figure 9. Since some arcs in the path may be directed from some v_i toward v_{i+1} and others from some v_{j+1} toward v_j, arrow heads have been omitted from the illustration.

Figure about here

Figure 9: A simple path linking s to t

By reordering the rows and the arcs in S, the system of equations corresponding to the arcs in the path may be illustrated as shown below.

$$\pm \begin{bmatrix} 1 \\ -1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix} y_1 \pm \begin{bmatrix} 0 \\ 1 \\ -1 \\ 0 \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix} y_2 \pm \begin{bmatrix} 0 \\ 0 \\ 1 \\ -1 \\ \vdots \\ 0 \\ 0 \\ 0 \end{bmatrix} y_3 \pm \cdots \pm \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 1 \\ -1 \end{bmatrix} y_q = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \\ -1 \end{bmatrix}$$

Since the first row of the above system has a single nonzero, y_1 is uniquely determined and must be either 1 or -1. Once y_1 is known, y_2 is then uniquely determined and must be either 1 or -1. Similarly, $y_3, \ldots, y_q$ can be determined successively. Therefore, a solution to $[S|e_\rho]y = e_{st}$ can be constructed by setting the components of y corresponding to the path from s to t in $< V, T >$ to ± 1s as described above and setting all other components to zero. This is generally called a *cycle trace* and is formally presented in the following procedure.

procedure CYCLE TRACE

inputs:	$[S\|e_\rho]$	-	$m \times m$ basis for a linear network problem
	e_{st}	-	m vector corresponding to a selected nonbasic arc

```
output:        $y$          -   $y = [S|e_p]^{-1}e_{st}$

begin
   [Initialization]
   let $< V,T >$ be the spanning tree corresponding to $S$;
   let $P = \{v_1, e_{i_1j_1}, v_2, e_{i_2j_2}, \ldots, v_q, e_{i_qj_q}, v_{q+1}\}$
   be the simple path in $< V,T >$ linking node $s$ to node $t$;
   for $i = 1, \ldots, m$
     $y_i = 0$;
   endfor
   $k \Leftarrow 1$;
   while $k \leq q$ do
     let $c$ be the column index of $S$ corresponding to arc $e_{i_kj_k}$;
     if $e_{i_kj_k} = e_{v_k} - e_{v_{k+1}}$
       then
         $y_c = 1$;
       else
         $y_c = -1$;
     endif
     $k \Leftarrow k + 1$;
   endwhile
end
```

Consider the basis tree in Figure 8 and suppose $s = 5$ and $t = 6$. The simple path and corresponding values of the updated column are illustrated in Figure 10. Note that the nonzero components of the updated column y are identical to the orientation function (see Section 1.7) on the simple path from s to t. Furthermore, the components of y are from $\{1, -1\}$.

Figure about here

Figure 10: Updated column example

3.4 Basis Exchange Update

As seen in Sections 3.2 and 3.3, the key operations for the primal simplex algorithm can be performed directly on the spanning tree $< V,T >$. In this section a data structure used to store the spanning tree in computer memory is presented along with an algorithm which will perform the basis exchange update using this data structure.

Suppose the rooted tree is drawn in the plane placing the root node at the top with the branches extending downward as illustrated in Figure 11a. One may imagine tracing a line around the contours of the tree as illustrated in Figure 11b. Traversing a tree in this way has become known as a depth-first search. For the example, the nodes in this search could be ordered as 4,3,4,1,5,1,2,1,4,7,6,7,4. By eliminating all duplicate occurances an ordering known as *preorder* is obtained. For this example, the corresponding preorder is 4,3,1,5,2,7,6. The label which gives the next node in the preorder for node v is known as the *thread*, denoted by $t(v)$. The thread for the example is illustrated in Figure 11c.

Figure about here

Figure 11: Illustration of thread labels for sample rooted tree

The other two labels are related to the path from a given node v to the root ρ in the basis tree $< V,T >$. Let

$$P = \{v_1, e_{i_1 j_1}, v_2, e_{i_2 j_2}, \ldots, v_q, e_{i_q j_q}, v_{q+1}\}$$

denote the simple path in $< V,T >$ which links v to ρ. The predecessor label $p(v)$ is v_2 (the second node on the path) and the distance label $d(v)$ is q (the number of arcs on the path). Both the predecessor and the distance labels of the root ρ are defined to be zero. The labels for the rooted tree shown in Figure 11 are given in Table 1.

Table 1. Labels for the rooted tree in Figure 11a

node	predecessor	thread	distance
v	$p(v)$	$t(v)$	d(v)
1	4	5	1
2	1	7	2
3	1	1	1
4	0	3	0
5	1	2	2
6	7	4	2
7	4	6	1

Figure about here

Figure 12: New rooted tree after basis exchange update

Both the dual calculation (Section 3.2) and the column update (Section 3.3) can be easily implemented in software using the triple labels to represent the basis tree $< V,T >$. The only tricky part is the technique needed for a basis exchange update. For example, suppose the arc $(2,7)$ is exchanged with the arc $(1,4)$ in the rooted tree illustrated in Figure 11.

Table 2. Labels after the basis exchange update

node v	predecessor $p(v)$	thread $t(v)$	distance $d(v)$
1	2	5	3
2	7	1	2
3	1	7	1
4	0	3	0
5	1	6	4
6	7	4	2
7	4	2	1

The new tree is illustrated in Figure 12 and the corresponding triple labels are given in Table 2. An algorithm which will perform this exchange is given below.

procedure BASIS EXCHANGE UPDATE

inputs: $< V,T >$ - the current basis tree
$p[i]$ - the predecessor label oi node i
$t[i]$ - the thread label of node i
$d[i]$ - the distance label of node i
(u,v) - the arc selected to become part of the new basis
k - node such that k and $p[k]$ are the end nodes of the arc which is to be removed from the current basis

outputs: $p[i]$ - updated predecessor label of node i
$t[i]$ - updated thread label of node i
$d[i]$ - updated distance label of node i

assumption: k and $p[k]$ are on the path from u to the root node of the current basis tree

```
begin
   [Initialization]
   $q \Leftarrow v$, $q' \Leftarrow v$, $i \Leftarrow q$, $j \Leftarrow p[q]$, $k' \Leftarrow p[k]$, $l \Leftarrow d[q'] + 1$, end $\Leftarrow$ 'no';
   while end = 'no' do
      $l' \Leftarrow d[i]$, $m \Leftarrow l - d[i]$, $d[i] \Leftarrow l$, $z \Leftarrow i$, $x \Leftarrow t[i]$;
      while $d[x] > l'$ do
         $d[x] \Leftarrow d[x] + m$, $z \Leftarrow x$, $x \Leftarrow t[x]$;
      endwhile
      $r \Leftarrow j$;
      while $t[r] \neq i$ do
      $r \Leftarrow t[r]$;
    endwhile
    $t[r] \Leftarrow x$;
    if $i = k$
      then
        $t[z] \Leftarrow t[q'], t[q'] \Leftarrow q, p[q] \Leftarrow q'$, end $\Leftarrow$ 'yes';
      else
        $t[z] \Leftarrow j, r \Leftarrow i, i \Leftarrow j, j \Leftarrow p[j], p[i] \Leftarrow r, l \Leftarrow l + 1$;
    endif
  endwhile
end
```

In our software implementation of the primal simplex algorithm for linear network problems, the basis exchange update is also integrated with the value update. In addition, since the set of nodes whose dual values change and the set of nodes whose distance labels change is the same set, the dual update calculation is also integrated with the basis exchange update. All of these specializations can result in software which can execute one hundred times faster than general purpose linear programming software. Most of the modern commercial linear programming systems now contain a specialized network solution module.

4 Generalized Networks

Let G be an $m \times n$ matrix with full row rank such that every column of G has at most two nonzero entries. Let c and u be n-component vectors and b be an m-component vector. Let $Y = \{x : Gx = b, \hat{0} \leq x \leq u\}$ and assume that $Y \neq \emptyset$. The *generalized network problem* is to find an n-component vector $\bar{x}$ such that

$$c\bar{x} = \min_{x}\{cx : x \in Y\}$$

Note that the linear network model is a special case of the generalized network problem. The revised model developed in Section 3.1 is simply

$$\min_{x}\{cx : (x, a) \in X^E\}$$

where

$$X^E = \{(x|a) : Ax + e_\rho a = r, \hat{0} \leq x \leq u\} \qquad .$$

The corresponding generalized model has $G = [A|e_\rho]$.

Let $V = \imath m = \{1, \ldots, m\}$. Then n arcs on the node set V can be constructed using the n columns of G. If $G_{.j}$ has two nonzero entries in rows i and k, we let the corresponding arc be (i, k). If $G_{.j}$ had only a single nonzero entry in row i, we let the corresponding arc be a root (i).

Consider the following matrix.

$$\begin{bmatrix} 1 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 3 & -2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 2 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 4 & 0 & -1 & 0 \\ 0 & 2 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 \end{bmatrix}$$

Using the rule presented above, the following network can be constructed:

$$< \{1,2,3,4,5,6,7\}, \{(1,2),(1,5),(2),(2,5),(3),(3,4),(3,6),(4,7),(6.7)\} >$$

A visual representation of this network is given in Figure 13. The numbers associated with the ends of the arcs correspond to the nonzero entries in the columns of G.

Figure about here

Figure 13: Example generalized network

4.1 Basis Characterization

Let G be an $m \times n$ matrix with full row rank such that every column of G has at most two nonzero entries. Let B be a basis for G and let $\bar{B}$ be a reordering of rows and columns of B such that $\bar{B}$ is displayed in block diagonal form. That is,

$$\bar{B} = \begin{bmatrix} \bar{B}^1 & & & \\ & \bar{B}^2 & & \\ & & \ddots & \\ & & & \bar{B}^p \end{bmatrix}$$

For example:

$$B = \begin{bmatrix} 1 & 2 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & -2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 2 & 2 & 0 \\ 0 & 0 & 0 & 0 & 4 & 0 & -1 \\ 0 & 2 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -1 \end{bmatrix}$$

and

$$\bar{B} = \left[\begin{array}{ccc|cccc} 1 & 0 & 2 & & & & \\ -1 & -2 & 0 & & 0 & & \\ 0 & 1 & 2 & & & & \\ \hline & & & 1 & 0 & 0 & 0 \\ & & & 0 & -1 & 0 & 0 \\ & 0 & & 0 & -1 & 4 & 0 \\ & & & 2 & 0 & 2 & 1 \end{array}\right] \begin{array}{c} (row) \\ 1 \\ 2 \\ 5 \\ 6 \\ 7 \\ 4 \\ 3 \end{array}$$

with $p = 2$. The number of blocks, p, can be as many as m. Let the networks associated with each of the blocks be denoted by $< V^1, E^1 >, \ldots, < V^p, E^p >$. For the above example, $< V^1 . E^1 > = < \{1,2,5\}, \{(1,2),(2,5),(1,5)\} >$ and $< V^2, E^2 > = < \{3,4,6,7\}, \{(3),(3,4),(3,6),(4,7)\} >$. The corresponding networks are illustrated in Figure 14

Figure about here

Figure 14: Basis for a generalized network

A connected network having exactly one cycle is traditionally called a *one-tree* and a connected network having exactly one root arc is traditionally called a *rooted tree*. It is well known that the connected components from a basis of a generalized network are either one-trees or rooted trees. That is, $< V^1, E^1 >, \ldots, < V^p . E^p >$ are either one-trees or rooted trees. For the basis illustrated in Figure 14, one component is a rooted tree and the other is a one-tree.

4.2 Dual Calculation

Since the basis $\bar{B}$ has special structure, $\pi \bar{B} = \bar{c}$ can be specialized to exploit this structure. Since $\bar{B}$ is block diagonal we obtain

$$[\ \pi^1 \mid \pi^2 \mid \cdots \mid \pi^p\ ] \begin{bmatrix} \bar{B}^1 & & & \\ & \bar{B}^2 & & \\ & & \ddots & \\ & & & \bar{B}^p \end{bmatrix} = [\ \bar{c}^1 \mid \bar{c}^2 \mid \cdots \mid \bar{c}^p\ ]$$

Hence, the p systems $\pi^q \bar{B}^q = \bar{c}^q$ cam be solved independently.

Two cases must be considered. Suppose $\bar{B}^q$ corresponds to a rooted tree. Then by row and column interchanges, $\bar{B}^q$ can be displayed in lower triangular form and $\pi^q \bar{B}^q = \bar{c}^q$ can be solved by back-substitution.

Consider the example illustrated in Figure 15 with

$$[\ \pi_6 \mid \pi_7 \mid \pi_4 \mid \pi_3\ ] \left[\begin{array}{c|c|c|c} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & -1 & 4 & 0 \\ 2 & 0 & 2 & 1 \end{array}\right] = [\ -1 \mid 2 \mid 4 \mid -2\ ]$$

Figure about here

Figure 15: Rooted tree component of a basis for a generalized network

From the fourth equation, $\pi_3 = -2$. Then from the third equation, $\pi_4 = 2$. The first two equations yield $\pi_6 = 3$ and $\pi_7 = -4$. For the rooted tree case, a procedure similar to the one given in Section 3.2 can be developed.

Figure about here

Figure 16: One-tree component of a basis for a generalized network

Consider the example illustrated in Figure 16 with

$$[\ \pi_1 \mid \pi_2 \mid \pi_5\ ] \left[\begin{array}{r|r|r} 1 & 0 & 2 \\ -1 & -2 & 0 \\ 0 & 1 & 2 \end{array} \right] = [\ 5 \mid 5 \mid 4\ ]$$

This system of equations for a cycle is almost lower triangular (only the last column has a nonzero entry above the diagonal) and takes the special form:

$$\pi \left[\begin{array}{c|c|c|c|c|c} a_1 & & & & & b_n \\ b_1 & a_2 & & & & \\ & b_2 & a_3 & & & \\ & & b_3 & \ddots & & \\ & & & \ddots & a_{n-1} & \\ & & & & b_{n-1} & a_n \end{array} \right] = c$$

which corresponds to the cycle illustrated in Figure 17. Under the assumption that the above system has full row rank, then the unique solution is given by:

$$\pi_1 = \frac{c_1/a_1 + w_1c_2/a_2 + w_1w_2c_3/a_3 + \cdots + w_1w_2\cdots w_{n-1}c_n/a_n}{1 - w_1w_2\cdots w_n} \tag{3}$$

and

$$\pi_{i+1} = \frac{c_i - a_i\pi_i}{b_i} \text{ for } i = 1, \ldots, n-1\,, \tag{4}$$

where $w_i = -b_i/a_i$ for $i = 1, \ldots, n$.

Figure about here

Figure 17: Cycle for a generalized network

For the example illustrated in Figure 17, $w_1 = 1, w_2 = 1/2, w_3 = -1, \pi_1 = 2, \pi_2 = -3$, and $\pi_3 = 0$.

An extension of the procedure DUAL CALCULATION given in Section 3.2 can be developed for the case of generalized networks. The basic idea is that the components corresponding to rooted trees are triangularizable and the components corresponding to one-trees are nearly triangularizable. For the components corresponding to one-trees, (3) and (4) are used for calculations involving the cycle and all other calculations involve a lower triangular system of equations. Hence, once the duals on the cycle are known, all other duals can be determined by back-substitution.

4.3 Column Update

Suppose that the nonbasic arc corresponding to G_j prices favorably and is selected for a potential flow change. The column update step of the primal simplex algorithm requires that y be determined where $y = \bar{B}^{-1}G_j$ or $\bar{B}y = G_j$. Since $\bar{B}$ is block diagonal, this may be written in the form:

$$\begin{bmatrix} \bar{B}^1 & & & \\ & \bar{B}^2 & & \\ & & \ddots & \\ & & & \bar{B}^p \end{bmatrix} \begin{bmatrix} y^1 \\ y^2 \\ \vdots \\ y^p \end{bmatrix} = \begin{bmatrix} g^1 \\ g^2 \\ \vdots \\ g^p \end{bmatrix}$$

Therefore, one must solve p systems of the form $\bar{B}^q y^q = g^q$ where the corresponding network $< V^q, E^q >$ is either a rooted tree or a one-tree. Furthermore, g^q will have at most two nonzero entries. If $g^q = \hat{0}$, then $y^q = \hat{0}$.

Suppose g^q has two nonzero entries a and b in positions i and j, respectively. Then $g^q = a\epsilon_i + be_j$. Let y_a^q and y_b^q solve $\bar{B}^q y_a^q = ae_i$ and $\bar{B}^q y_b^q = be_j$, respectively. Then $y^q = y_a^q + y_b^q$. Hence, a specialized algorithm for the column update can be developed from an algorithm to efficiently solve the system $\bar{B}^q y_c^q = ce_k$ where $< V^q, E^q >$ is either a rooted tree or a one-tree.

If $< V^q, E^q >$ is a rooted tree, then $\bar{B}^q$ is triangularizable and y_c^q can be found by forward substitution. Consider the example illustrated in Figure 18 with

$$\left[\begin{array}{c|c|c|c} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & -1 & 4 & 0 \\ 2 & 0 & 2 & 1 \end{array}\right] \begin{bmatrix} y_{36} \\ y_{47} \\ y_{34} \\ y_{30} \end{bmatrix} = \begin{bmatrix} 0 \\ 5 \\ 0 \\ 0 \end{bmatrix}$$

From the first equation, $y_{36} = 0$. From the second equation, $y_{47} = -5$. From the third equation, $y_{34} = -5/4$ and from the fourth equation, $y_{30} = 5/2$.

Consider the one-tree illustrated in Figure 19 with

$$\left[\begin{array}{c|c|c} 1 & 0 & 2 \\ -1 & -2 & 0 \\ 0 & 1 & 2 \end{array}\right] \begin{bmatrix} y_{12} \\ y_{25} \\ y_{15} \end{bmatrix} = \begin{bmatrix} 5 \\ 0 \\ 0 \end{bmatrix}$$

Figure about here

Figure 18: Column update for a rooted tree component of a basis for a generalized network

The system of equations for a cycle is almost lower triangular (only the last column has a nonzero entry above the diagonal) and takes the special form:

$$\left[\begin{array}{c|c|c|c|c|c} a_1 & & & & & b_n \\ b_1 & a_2 & & & & \\ & b_2 & a_3 & & & \\ & & b_3 & \ddots & & \\ & & & \ddots & a_{n-1} & \\ & & & & b_{n-1} & a_n \end{array}\right] y = \alpha e_1$$

which corresponds to the cycle illustrated in Figure 17. Under the assumption that the above system has full row rank, the unique solution is given by:

$$y_1 = \frac{\alpha / a_1}{1 - w_1 w_2 \cdots w_n} \qquad (5)$$

and

$$y_{i+1} = \frac{-b_i y_i}{a_{i+1}} \text{ for } i = 1, \ldots, n-1\,, \qquad (6)$$

where $w_i = -b_i/a_i$ for $i = 1, \ldots, n$.
For the example illustrated in Figure 19, $w_1 = 1, w_2 = 1/2, w_3 = -1, y_{12} = 10/3, y_{25} = -5/3$, and $y_{15} = 5/6$.

Figure about here

Figure 19: Column update for a one-tree component of a basis for a generalized network

An extension of the procedure cycle trace given in Section 3.3 can be developed for the case of generalized networks. The basic idea is that the rooted trees are triangularizable and the one-trees are nearly triangularizable. For the components corresponding to one-trees, (5) and (6) are used for calculations involving a lower triangular system of equations. If the nonbasic column of interest, $G_{.j}$, has two nonzero entries, then the procedure is applied twice and the results are added.

4.4 A Data Structure

The data structure presented in Section 3.4 has been extended to accomodate the one-tree components of a generalized network basis. With respect to the distance label. the nodes in the cycle of a one-tree and the root node in a rooted tree are treated in the same way with all nodes in a cycle having distance label of zero. The predecessor label of nodes in a cycle point to the next node in the cycle. Beginning with any node in a cycle, say v, the sequence $v, p(v), p(p(v)), \ldots$ will identify all nodes in the cycle and will eventually return to node v. For root nodes. v we adopt the convention $p(v) = v$. The thread has also been extended in an obvious way with the convention that traversal around a cycle using the thread is in the opposite direction to that using the predecessor. The data structure corresponding to the generalized basis illustrated in Figure 20 is given in Table 3.

Figure about here

Figure 20: Basis for a generalized network with thᴾ predecessor and thread labels illustrated

Table 3. Data structure for the basis illustrated in Figure 20

node	predecessor	thread	distance	multipliers	
v	$p(v)$	$t(v)$	$d(v)$	a	b
1	5	2	0	9	8
2	1	11	0	2	1
3	3	6	0	5	0
4	3	7	1	6	7
5	2	8	0	4	3
6	3	4	1	10	11
7	4	3	2	12	13
8	5	9	1	14	15
9	8	10	2	16	17
10	8	1	2	18	19
11	2	12	1	20	21
12	2	2	1	22	23

Efficient procedures have been developed for updating this data structure after a basis exchange. As with the linear network case, the flow or value update and the dual variable update is usually integrated with the basis exchange update. All of these specialized techniques can result in software from one to two order of magnitude faster than general linear programming software. All calculations involving the flows, dual variables, and updated columns must be performed using real arithmetic as opposed to the linear network case which only requires addition and subtraction of integers when the data is integral.

Some additional computational efficiencies can be achieved by scaling columns of G so that every column has at least one +1 entry. The disadvantage of using this scaling is that the resulting software cannot be easily incorporated within a branch-and-bound algorithmn which can accomodate integrality restrictions on some or all of the flow variables.

5 Multicommodity Networks

Multicommodity networks arise in practice when more than one type of commodity must share the capacity of the arcs in a network. Typical examples include Air Force cargo routing models in which cargo with different origin-destination pairs must share the capacity of the various aircraft which represent arcs in the model. For this model, a commodity represents all cargo with a given base of origin. The well-known LOGAIR Model is a model of this type.

Another well-known Air Force model is the family of Patient Distribution System (PDS) Models. The PDS generator was developed to assist in evaluating the airlift capacity of moving patients from a European theatre to U.S. hospitals. One of the input parameters for the PDS generator is the number of days in the model. A 20 day model has over 100,000 columns in the corresponding linear program.

Let $G =< V, E >$ be the underlying network through which K commodities will be flowing and let the $m \times n$ matrix A denote the corresponding node-arc incidence matrix. Let the m-component vector r^k denote the requirements vector for commodity k and the n-component vector x^k denote the flows for commodity k with corresponding unit costs and upper bounds of c^k and u^k, respectively. Let $Y^k = \{x^k : Ax^k = r^k \, , \, \hat{0} \leq x^k \leq u^k\}$ and assume that $Y^k \neq \emptyset$. Suppose the shared (also called mutual) capacity for arc i is b_i, an entry of the n-component vector b. The multicommodity network flow problem is to find K n-component vectors $\bar{x}^1, \ldots, \bar{x}^K$ such that

$$\sum_k c^k \bar{x}^k = \min\{\sum_k c^k x^k \; : \; \sum_k x^k \leq b \, , \, x^k \in Y^k\}$$

It is possible to generalize the multicommodity network model to allow for both commodity dependent networks ($A^k x^k = r^k$) and for multipliers on the mutual capacity constraints ($\sum_k D^k x^k \leq b$, where each D^k is an $n \times n$ diagonal matrix with nonnegative entries). It is also common that the mutual capacity constraints do not involve all n of the arcs. These generalizations present no mathematical difficulties in the algorithms to follow but greatly complicate the notation.

Consider the sample two commodity network illustrated in Figure 21. The matrices and vectors corresponding to this model are as follows:

$$A = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ -1 & 0 & 1 & 1 & 0 \\ 0 & -1 & -1 & 0 & 1 \\ 0 & 0 & 0 & -1 & -1 \end{bmatrix}$$

$$r^1 = [\ 5 \quad 0 \quad 0 \quad -5\ ]$$

$$r^2 = [\ 4 \quad 2 \quad -3 \quad -3\ ]$$

$$c^1 = [\ 1 \quad 3 \quad 5 \quad 7 \quad 9\ ]$$

$$c^2 = [\ 2 \quad 4 \quad 6 \quad 8 \quad 0\ ]$$

$$u^1 = [\ 4 \quad 2 \quad 1 \quad 3 \quad 3\ ]$$

$$u^2 = [\ 2 \quad 3 \quad 2 \quad 2 \quad 3\ ]$$

$$b = [\ 6 \quad 3 \quad 6 \quad 5 \quad 3\ ]$$

Figure about here

Figure 21: Example multicommodity (two commodity) network model

After adding slack variables to the mutual capacity constraints the system of equations corresponding to this model is

$$M \begin{bmatrix} x^1 \\ x^2 \\ s \end{bmatrix} = \begin{bmatrix} r^1 \\ r^2 \\ b \end{bmatrix}$$

where M is the matrix

$$\left[\begin{array}{rrrrr|rrrrr|rrrrr}
1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & -1 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\hline
0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 \\
\hline
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1
\end{array}\right]$$

In general, the linear programming constraint matrix corresponding to a multicommodity network model takes the form:

$$\begin{pmatrix} A & & & & \\ & A & & & \\ & & \ddots & & \\ & & & A & \\ I & I & \cdots & I & I \end{pmatrix} \tag{7}$$

Hence, the columns either take the form e_i for some i or have exactly three nonzero entries (two +1s and one −1).

There are three types of specialized algorithms which have been used to exploit the underlying structure of the multicommodity model. *primal partitioning, price-directive decommposition*, and *resource-directive decomposition*. Primal partitioning is a specialization of the primal simplex method which exploits the network structure of the basis. Price-directive decomposition is a specialization of Dantzig-Wolfe decomposition which exploits the network structure of the subproblems. Resource-directive decomposition allocates the mutual capacity among the K commodities and solves K linear network problems per allocation. The trick is to develop a reallocation scheme which will guarantee that an optimum can be found.

5.1 Primal Partitioning

By appending a root arc to the network for each of the K commodities, we convert the matrix (7) into a matrix having full row rank. That is, the full rank linear programming constraint matrix takes the form:

$$\begin{array}{c} \\ m \\ m \\ \vdots \\ m \\ n \end{array} \begin{array}{c} \begin{array}{ccccc} n+1 & n+1 & & n+1 & n \end{array} \\ \begin{pmatrix} A|e_p & & & & \\ & A|e_p & & & \\ & & \ddots & & \\ & & & A|e_p & \\ I|0 & I|0 & \cdots & I|0 & I \end{pmatrix} \end{array} \tag{8}$$

where A is a node-arc incidence matrix. It is well known that every basis for (8) may be partitioned in the following form:

$$\begin{array}{c} m \\ \vdots \\ m \\ q \\ n-q \end{array} \begin{pmatrix} B^1 & & & R^1 & & & \\ & \ddots & & & \ddots & & \\ & & B^K & & & R^K & \\ P^1 & \cdots & P^K & T^1 & \cdots & T^K & \\ S^1 & \cdots & S^K & U^1 & \cdots & U^K & I \end{pmatrix}$$

where $B^1, \ldots, B^K$ correspond to rooted trees. This basis takes the general form:

$$\begin{array}{c|ccc} & mK & n-p & p \\ \hline mK & L_1 & R_1 & \\ q & L_2 & R_2 & \\ p & L_3 & R_3 & I \end{array} \tag{9}$$

where

$$L_1 = \begin{pmatrix} B^1 & & \\ & \ddots & \\ & & B^K \end{pmatrix}$$

$$R_1 = \begin{pmatrix} R^1 & & \\ & \ddots & \\ & & R^K \end{pmatrix}$$

$$L_2 = [\ P^1 \ \cdots \ P^K \ ]$$

$$R_2 = [\ T^1 \ \cdots \ T^K \ ]$$

$$L_3 = [\ S^1 \ \cdots \ S^K \ ]$$

and

$$R_3 = [\ U^1 \ \cdots \ U^K \ ]$$

By partitioning the dual variables to be compatible with (9), the dual calculation requires the solution to the following system:

$$[\ \pi_1 \mid \pi_2 \mid \pi_3 \ ] \left[\begin{array}{c|c|c} L_1 & R_1 & \\ \hline L_2 & R_2 & \\ \hline L_3 & R_3 & I \end{array} \right] = [\ c_1 \mid c_2 \mid 0 \ ]$$

Let $Q = [R_2 - L_2 L_1^{-1} R_1]$. Then $\pi_3 = \hat{0}$, $\pi_2 = (c_2 - c_1 L_1^{-1} R_1) Q^{-1}$, and $\pi_1 = (c_1 - \pi_2 L_2) L_1^{-1}$. Note that L_1 corresponds to K rooted trees so that all calculatiions involving L_1^{-1} can be executed directly on the rooted trees and do not require the use of explicit matrices. The only matrix operations involve the $q \times q$ matrix Q which is called the ***working basis***.

Partitioning the updated column in a similar way, the column update calculation requires the solution of the following system:

$$\left[\begin{array}{c|c|c} L_1 & R_1 & \\ L_2 & R_2 & \\ L_3 & R_3 & I \end{array} \right] \left[\begin{array}{c} x \\ y \\ z \end{array} \right] = \left[\begin{array}{c} u \\ t \\ w \end{array} \right]$$

The solution is given by $y = Q^{-1}(r - L_2L_1^{-1}u)$, $x = L_1^{-1}u - L_1^{-1}R_1y$, and $z = w - L_3L_1^{-1}u - (R_3 - L_3L_1^{-1}R_1)y$. As before, the only matrix operations ir.volve the working basis Q.

Each basis exchange can alter the dimension of the working basis by at most one. Efficient techniques have been developed for maintaining Q^{-1} as the basis is updated. If q is small, then most of the work associated with a simplex pivot can be performed efficiently using the K rooted trees. If q is large, then the pivots can be more expensive than a regular simplex pivot which does not exploit the special network structure.

5.2 Price-Directive Decomposition

For each $k \in \imath K$, suppose that $Y^k = \{x^k : Ax^k = r^k, \hat{0} \leq x^k \leq u^k\} \neq \emptyset$ and is bounded, and let $\bar{w}_1^k, \ldots, \bar{w}_{n_k}^k$ denote the extreme points of Y^k. Let the $n \times n_k$ matrix W^k be given by $[\ \bar{w}_1^k \mid \bar{w}_2^k \mid \cdots \mid \bar{w}_{n_k}^k\ ]$. Then any $\bar{x}^k \in Y^k$ can be represented as a convex combination of those extreme points (the columns of W^k). That is, $\bar{x}^k = W^k\lambda^k$ where $\hat{1}\lambda^k = 1$ and $\lambda^k \geq \hat{0}$. Let

$$W = \{(\ \lambda^1 \mid \cdots \mid \lambda^k \mid s\ ) : \textstyle\sum_k W^k\lambda^k + s = b,\ s \geq \hat{0},\ \hat{1}\lambda^k = 1,\ \lambda^k \geq \hat{0}\}$$

Then an equivalent statement of the multicommodity network problem is to find $K + 1$ vectors $(\ \bar{\lambda}^1 \mid \cdots \mid \bar{\lambda}^k \mid \bar{s}\ )$ such that

$$\textstyle\sum_k c^k W^k \bar{\lambda}^k = \min\{\sum_k c^k W^k \lambda^k : (\ \lambda^1 \mid \cdots \mid \lambda^k \mid s\ ) \in W\}$$

The disadvantage of this statement of the problem is that it is quite difficult to find all the extreme points for Y^k (which form the columns of W^k). However, the price-directive algorithm provides a mechanism for applying the primal simplex algorithm to this formulation by providing columns of W^k only when they are needed. In fact, the complete matrix W^k is never actually generated.

Suppose we have a nonnegative basic feasible solution to the system of equations:

$$\begin{aligned}
\sum_k W^k\lambda^k + s &= b && (\pi) \\
\hat{1}\lambda^1 &= 1 && (\gamma_1) \\
&\vdots \\
\hat{1}\lambda^K &= 1 && (\gamma_K)
\end{aligned}$$

with the corresponding duals $(\ \pi \mid \gamma_1 \mid \cdots \mid \gamma_k\ )$. From Section 2, we see that the ith nonbasic slack variable prices favorably if $\pi_i > 0$. Also, λ_j^k prices favorably if $\bar{w}_j^k\pi + \gamma_k - c^k\bar{w}_j^k > 0$. Note that the extreme point from Y^k which prices most favorably can be found by solving the linear network problem

$$\min\{(c^k - \pi)u^k \ : \ w^k \in Y^k\}$$

using an extreme point algorithm. If the extreme point produced from solving the linear network problem prices unfavorably, then none of the columns of W^k will price favorably. Hence, columns of W^k are generated only as they are needed.

The linear program

$$\min\{\sum_k c^k W^k \lambda^k \ : \ (\ \lambda^1 \mid \cdots \mid \lambda^k \mid s\ ) \in W\}$$

is known as the *master problem* and the linear network problems $\min\{c^k x^k \ : \ x^k \in Y^k\}$ for $k \in \imath K$ are called the *slave subproblems*. Columns of the master problem (extreme points of the Y^k) are found by solving linear network problems.

A mathematical description of the price-directive algorithm follows:

```
procedure PRICE-DIRECTIVE DECOMPOSITION

inputs:      $A$                   -  $m \times n$ node-arc incidence matrix
             $K$                   -  number of commodities
             $c^1, \ldots, c^K$    -  $n$-component vectors of unit costs
             $u^1, \ldots, u^K$    -  $n$-component vectors of upper bounds
             $r^1, \ldots, r^K$    -  $m$-component vectors of requirements
             $b$                   -  $n$-component vector of mutual capacities

outputs:     $\bar{y}^1, \ldots, \bar{y}^K$  -  $n$-component vectors of optimal flows
assumption:  $Y^k = \{x^k : Ax^k = r^k, \hat{0} \le x^k \le u^k\} \ne \Phi$ and is bounded
             for all $k \in \imath K$.
begin
  [Initialize Master Problem]
  obtain a basic feasible solution $[\bar{y}^1 | \ldots | \bar{y}^K]$ for $W$ with corresponding dual variables
  $[\pi | \gamma]$ (if a basic feasible solution is not readily available, then artificial variables and
  a two-phase procedure may be used);
  optimal $\Leftarrow$ 'no';
  while optimal = 'no' do
    [Price Slacks]
    $i \Leftarrow 1$;
    while $i \le n$ do
      if $\pi_i \le 0$
        then
          perform one simplex iteration in the master problem using $[e_i | \hat{0}]$ as
          the nonbasic column which prices favorably;
          update $[\bar{y}^1 | \ldots | \bar{y}^K]$ and $[\pi | \gamma]$;
          $i \Leftarrow 1$;
        else
          $i \Leftarrow i + 1$;
      endif
    endwhile
    [Price Extreme Points]
    $k \Leftarrow 1$, favorable $\Leftarrow$ 'no';
    while $k \leq K$ and favorable = 'no'do
      obtain an optimal extreme point $w^k$ for $\min\{(c^k - \pi)x^k : Ax^k =$
      $r^k, \bar{0} \leq x^k \leq u^k\}$;
      if $(c^k - \pi)x^k \leq \gamma_k$
        then
          favorable $\Leftarrow$ 'yes';
        else
          $k \Leftarrow k + 1$;
      endif
    endwhile
    if favorable = 'yes'
      then
        perform one primal simplex iteration in the master problem using
        $[\bar{u}^k|e_k]$ as the nonbasic column which prices favorably;
        update $[\bar{y}^1|\ldots|\bar{y}^K]$ and $[\pi|\gamma]$;
      else
        optimal = 'yes';
    endif
  endwhile
end
```

A pictorial description of the price-directive decomposition algorithm is given in Figure 22. The master program produces dual variables used to construct the objective function coefficients for each of the slave subproblems. Each slave subproblem, in turn, produces an extreme point which may or may not provide an objective-improving column relative to the current basis for the master problem. A favorable column will be used for a simplex iteration in the master problem, producing new dual variables for the slave subproblems. Note that successive calls to a routine which solves a given subproblem involve only a change in the objective function and do not involve any changes to the constraints. Hence each such call can make use of the optimal basis tree produced by the previous call. We also recommend that a wrap-around list be maintained to organize the calls to the subproblems, a minor variation on the above procedure. That is, if commodity k produced a favorably priced column at iteration i, then begin the new iteration by pricing the columns associated with commodity $(k+1)\text{mod}K$.

Figure about here

Figure 22: The price-directive decomposition algorithm

5.3 Resource-Directive Decomposition

It is the mutual capacity constraints $\sum_k x^k \leq b$ that make the multicommodity problem difficult. If these constraints could be eliminated or ignored, then the problem would decompose into K linear network problems. Let $[\ \bar{y}^1 \mid \cdots \mid \bar{y}^K\ ]$ denote an allocation of b among the K commodities such that $\hat{0} \leq \bar{y}^k \leq u^k$ for each k and $\sum_k \bar{y}^k \leq b$. Let

$$z^k(\bar{y}^k) = \min\{c^k x^k \ :\ Ax^k = r^k\ ,\ \hat{0} \leq x^k \leq \bar{y}^k\}$$

An equivalent statement of the multicommodity network problem is to find K vectors $[\ \bar{y}^1 \mid \cdots \mid \bar{y}^K\ ]$ such that

$$\sum_k z^k(\bar{y}^k) = \min\{\sum_k c^k x^k \ :\ Ax^k = r^k\ ,\ \hat{0} \leq x^k \leq \bar{y}^k\}$$

The objective function $g([\ y^1 \mid \cdots \mid y^K\ ]) = \sum_k z^k(y^k)$ is piece-wise linear and convex. Hence, the classic subgradient algorithm can be applied to this problem.

Consider the nonlinear program $\min\{g(y)\ :\ y \in Y\}$, where $g(\cdot)$ is piece-wise linear and convex and $Y \neq \emptyset$ is formed by the intersection of a finite number of closed half-spaces. To apply the subgradient algorithm, one must be able to solve the problem

$$\min\{[\sum_i (y_i - \bar{p}_i)^2]^{1/2}\ :\ y \in Y\}$$

for any point $\bar{p}$. The solution of this problem is called the *projection* of $\bar{p}$ onto Y. This is traditionally expressed as $\bar{y} \longleftarrow P[\bar{p}]$.

Mathematically the subgradient algorithm may be stated as follows:

```
                        procedure SUBGRADIENT

 inputs:        $g(\cdot)$           -  the objective function
                $Y$                  -  the feasible region
                $s_1, s_2, \ldots$   -  the sequence of step size values
 output:        $\bar{y}$            -  an optimum for $\min\{g(y) : y \in Y\}$
 assumptions:   $g(\cdot)$ is piece-wise linear and convex
                $Y \neq \Phi$ is formed by the intersection of a finite number of
                closed half-spaces
begin
  [Initialization]
  obtain a point $\bar{y} \in Y$, optimal $\Leftarrow$ 'no', $i \Leftarrow 1$;
  while optimal = 'no' do
    obtain a subgradient $\bar{v}$ of $g(\cdot)$ at $\bar{y}$;
    if $\bar{v} \neq \hat{0}$
      then
        optimal $\Leftarrow$ 'yes';
      else
        $\bar{y} \Leftarrow P[\bar{y} - s_i \bar{v}]$, $i \Leftarrow i + 1$;
    endif
  endwhile
end
```

Several convergence results for this algorithm have appeared in the literature. These results all provide restrictions and guidance on the selection of the sequence of step sizes.

The computationally expensive part of the above algorithm when applied to the multicommodity network flow problem is the calculation of the subgradient. That is, we need a subgradient of the function $g([\ y^1 \mid \cdots \mid y^K\ ])$ at the point $[\ \bar{y}^1 \mid \cdots \mid \bar{y}^K\ ]$, where $g([\ \bar{y}^1 \mid \cdots \mid \bar{y}^K\ ]) = \sum_k z^k(\bar{y}^k)$. By duality theory,

$$\begin{aligned} z^k(\bar{y}^k) &= \min\{c^k x^k : Ax^k = r^k, \hat{0} \leq x^k \leq \bar{y}^k\} \\ &= \max\{r^k \mu^k - \bar{y}^k \nu^k : \mu^k A - \nu^k \leq c^k, \nu^k \geq \hat{0}\} \end{aligned}$$

Let $[\bar{\mu}^k | \bar{\nu}^k]$ for all $k \in \imath K$ denote optimal dual variables for $g([\ \bar{y}^1 \mid \cdots \mid \bar{y}^K\ ])$. It can easily be shown that $[\ -\bar{\nu}^1 \mid \cdots \mid -\bar{\nu}^K\ ]$ is a subgradient for $g(\cdot)$ at the point $([\ \bar{y}^1 \mid \cdots \mid \bar{y}^K\ ])$. Efficient projection algorithms for the special case of the multicommodity network problem are also available in the literature.

The major attraction of the subgradient algorithm is that the multicommodity network problem can be solved by solving a sequence of single commodity network problems. The major disadvantage is that it is quite difficult to select a set of step sizes so that the software implementation is robust over a wide range of problems. Our experience has indicated that much skill is required in the selection of the step sizes. Many skilled mathematical programming software developers have discovered that the step size selection is more of an art than a science.

6 Reference Notes

The specialized algorithms presented in this chapter all rely on the graphical interpretation of the simplex steps when applied to a linear program possessing an underlying network structure, in whole or in part. Graphical characterizations for bases for both the linear network problem and the generalized network problem can be found in Dantzig [1963]. Additional elaboration on the interpretation of the algebraic operations on a graphical structure were provided by

Johnson [1965]. The first software implementations which empirically demonstrated the merit of those specialized algorithms were developed by Srinivasan and Thompson [1973] and by Glover and Klingman and their colleagues at the University of Texas (see Glover, Karney, and Klingman [1974], Glover, Karney, Klingman, and Napier [1974], and Glover, Klingman, and Stutz [1974]). Since these seminal papers, hundreds of papers with various extensions and specializations have appeared in the literature. Bradley, Brown, and Graves [1977] and Barr, Glover, and Klingman [1979] are two of our favorites.

The first specialized primal simplex code which exploited the graphical nature of the basis of a generalized network was developed by Glover, Klingman, and Stutz [1973]. Since then, many excellent software implementations have been developed. A table of codes and references may be found in Clark, Kennington, Meyer, and Ramamurti [1992]. One of our favorite generalized network codes is GENNET which was developed by Brown and McBride [1984].

The primal partitioning method for multicommodity problems is due primarily to Hartman and Lasdon [1970,1972]. The price-directive decomposition procedure was first developed by Ford and Fulkerson [1958]. It is purported that the well-known Dantzig-Wolfe decomposition algorithm (see Dantzig and Wolfe [1960]) was inspired by the Ford and Fulkerson paper on multicommodity problems. The subgradient algorithm was first applied to the multicommodity network problem by Held, Wolfe, and Crowder [1974].

In this chapter, the algorithms presented were generally based on a specialization of the simplex method. There are three other basic approaches which are now competing quite successfully in empirical analyses. The oldest is the relaxation method which was developed by Bertsekas and his colleagues at MIT (see Bertsekas [1991]). The second is the algorithm of Goldberg which has been implemented in software by Anderson and Setubal [1992]. The third algorithm which could have an impact on the field is the network interior point method of Resende and Veiga [1992]. All of these approaches have their advocates and we expect improvements for each of these relatively new algorithms to appear in the literature in the near future.

Many books exist which contain excellent presentations of the various algorithms for a variety of network problems. Our favorites include Papadimitriou and Steiglitz [1982], Chvátal [1983], and Ahuja, Manganti, and Orlin [1989]. This chapter would not be complete without mentioning our own book (Kennington and Helgason [1980]) which contains a wealth of technical details not found in other publications.

7 References

Ahuja, R., T. Magnanti, and J. Orlin, [1989], "Chapter IV: Network Flows," in G. L. Nemhauser, A. H. G. Rinnooy Kan, and M. J. Todd (eds.), *Handbooks in Operations Research and Management Science: Volume 1 Opti-*

mization, North-Holland Publishing Company, Amsterdam, The Netherlands.

Anderson, R. J. and J. C. Setubal, [1992], "Goldberg's Algorithm for Maximal Flow in Perspective: A Computational Study," in D. S. Johnson and C. C. McGeoch (eds.), *DIMACS Implementation Challenge Workshop - Algorithms for Network Flows and Matching*, forthcoming.

Barr, R., F. Glover, and D. Klingman, [1979], "Enhancement of Spanning Tree Labeling Procedures for Network Optimization," *INFOR*, 17, 16-34.

Bertsekas, D. P., [1991], *Network Optimization: Algorithms and Codes*, The MIT Press, Cambridge, MA.

Bradley, G., G. Brown, and G. Graves, [1977], "Design and Implementation of Large Scale Primal Transshipment Algorithms," *Management Science*, 21, 1-38.

Brown, G. and R. McBride, [1984], "Solving Generalized Networks," *Management Science*, 30, 1497-1523.

Chvátal, V., [1983], *Linear Programming*, W. H. Freeman and Company, New York, NY.

Clark, R., J. Kennington, R. Meyer, and M. Ramamurti, [1992]. "Generalized Networks: Parallel Algorithms and an Empirical Analysis," *ORSA Journal on Computing*. 4, 132-145.

Dantzig. G. B., [1963], *Linear Programming and Extensions*. Princeton University Press, Princeton, NJ.

Dantzig, G. B., and P. Wolfe, [1960], "Decomposition Principle for Linear Programs," *Operations Research*, 8, 101-111.

Ford. L. R., and D. R. Fulkerson, [1958], "A Suggested Computation for Maximal Multicommodity Network Flow." *Management Science*, 5, 97-101.

Glover. F., D, Karney, and D. Klingman, [1974], "Implementation and Computational Comparisons of Primal, Dual, and Primal-Dual Computer Codes for Minimum Cost Network Flow Problems," *Networks*, 4, 191-212.

Glover, F., D, Karney, D. Klingman, and A. Napier, [1974], "A Computational Study on Start Procedures, Basis Change Criteria, and Solution Algorithms for Transportation Problems," *Management Science*. 20, 793-813.

Glover, F., D. Klingman. and J. Stutz, [1973], "Extension of the Augmented Predecessor Index Method to Generalized Network Problems," *Transportation Science*, 7, 377-384.

Glover, F., D. Klingman, and J. Stutz, [1974], "Augmented Threaded Index Method for Network Optimization," *INFOR*, 12, 293-298.

Hartman, J. K., and L. S. Lasdon, [1970], "A Generalized Upper Bounding Method for Doubly Coupled Linear Programs," *Naval Research Logistics Quarterly*, 17, 4, 411-429.

Hartman. J. K.. and L. S. Lasdon, [1972], "A Generalized Upper Bounding Algorithm for Multicommodity Network Flow Problems," *Networks*, 1, 333-354.

Held, M., P. Wolfe, and H. Crowder, [1974], "Validation of Subgradient Optimization," *Mathematical Programming*, 6, 62-88.

Johnson, E., [1965], "Programming in Networks and Graphs," Technical Report ORC 65-1, Operations Research Center, University of California-Berkeley, Berkeley, CA.

Kennington, J. L., and R. V. Helgason, [1980], *Algorithms for Network Programming*, John Wiley and Sons, Inc., New York, NY.

Papadimitriou, C. and K. Steiglitz, [1982], *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Resende, M. and G. Veiga, [1992], "An Efficient Implementation of a Network Interior Point Method," Technical Report, AT&T Bell Laboratories, Murray Hill, NJ 07974.

Srinivasan. V., and G. Thompson, [1973], "Benefit-Cost Analysis of Coding Techniques for the Primal Transportation Algorithm," *Journal of the Association for Computing Machinery*, 20, 194-213.

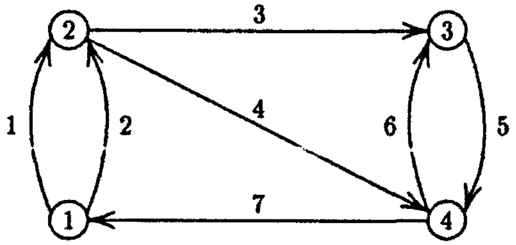
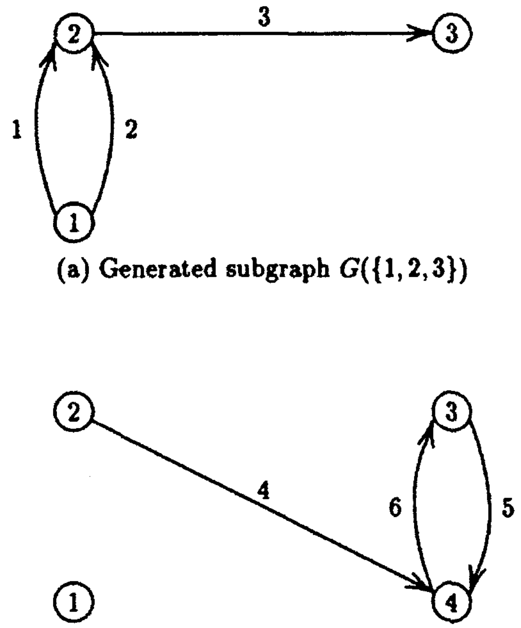


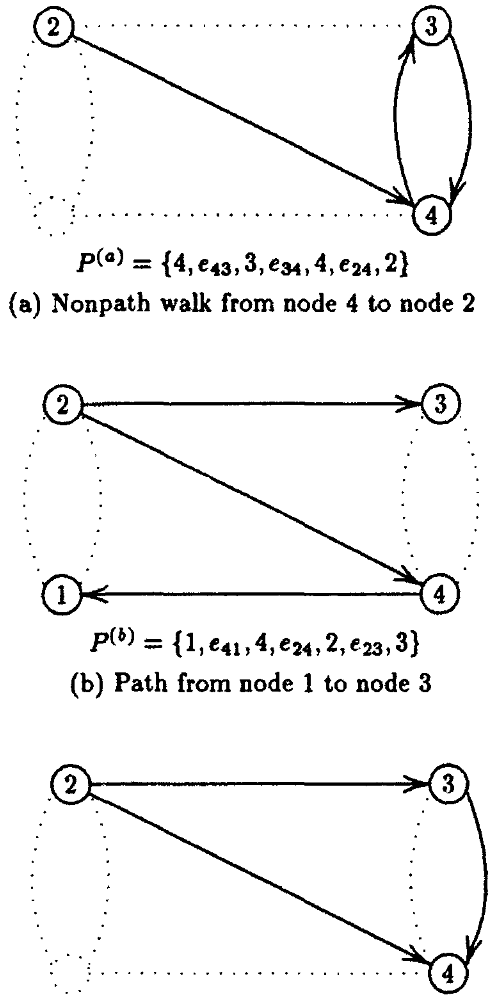
Figure 1: Example network



(a) Generated subgraph $G(\{1,2,3\})$

(b) A spanning subgraph for G

Figure 2: Example network subgraphs



$P^{(a)} = \{4, e_{43}, 3, e_{34}, 4, e_{24}, 2\}$

(a) Nonpath walk from node 4 to node 2

$P^{(b)} = \{1, e_{41}, 4, e_{24}, 2, e_{23}, 3\}$

(b) Path from node 1 to node 3

$P^{(c)} = \{2, e_{23}, 3, e_{34}, 4, e_{24}, 2\}$

(c) A cycle on nodes $\{2, 3, 4\}$

Figure 3: Walks in the example network

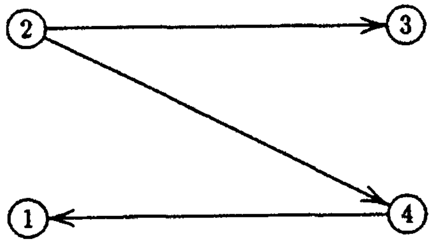


Figure 4: Example tree

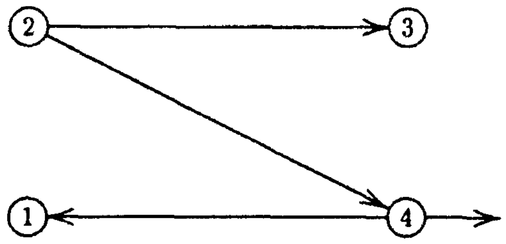


Figure 5: Example rooted tree

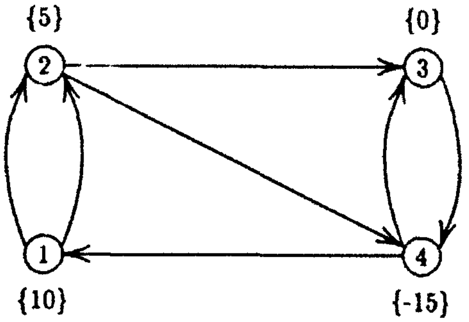
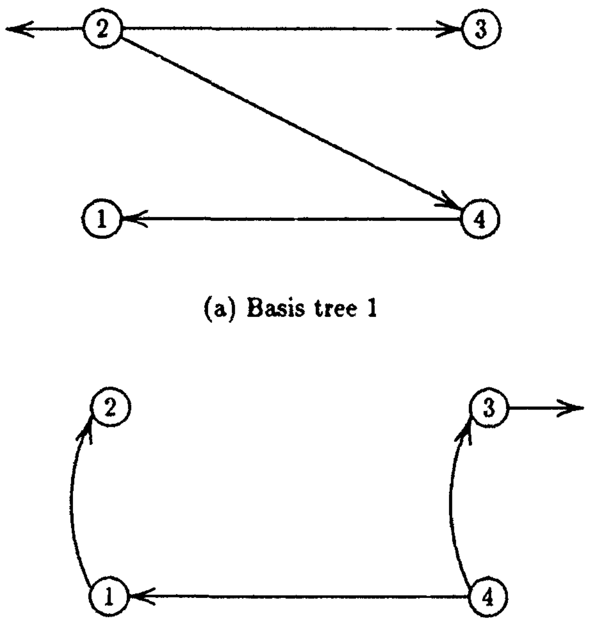


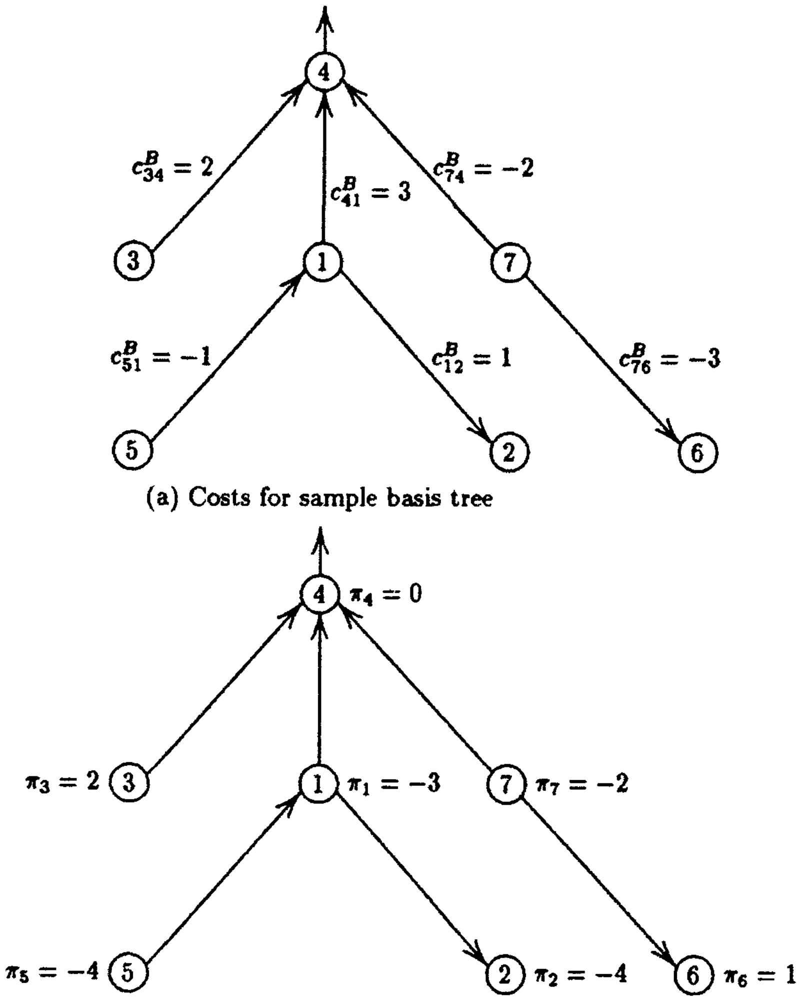
Figure 6: Example network with requirements



(a) Basis tree 1

(b) Basis tree 2

Figure 7: Basis trees for sample network



(a) Costs for sample basis tree

(b) Dual variables for sample basis tree

Figure 8: Dual variable calculation example

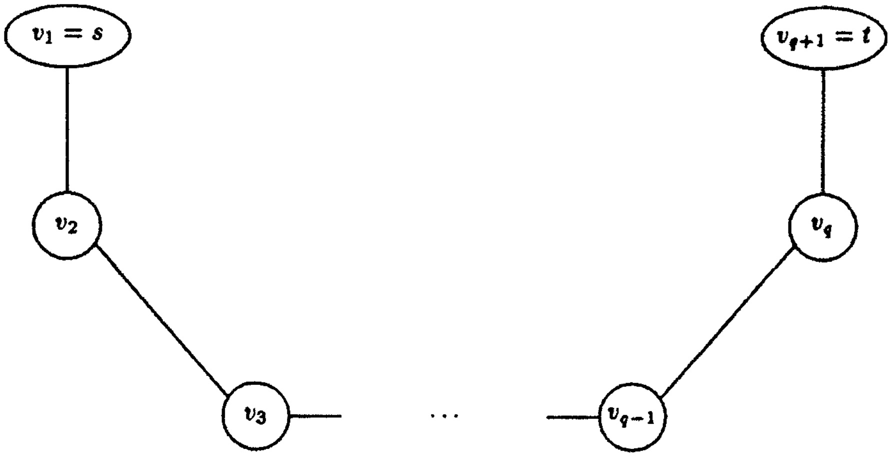
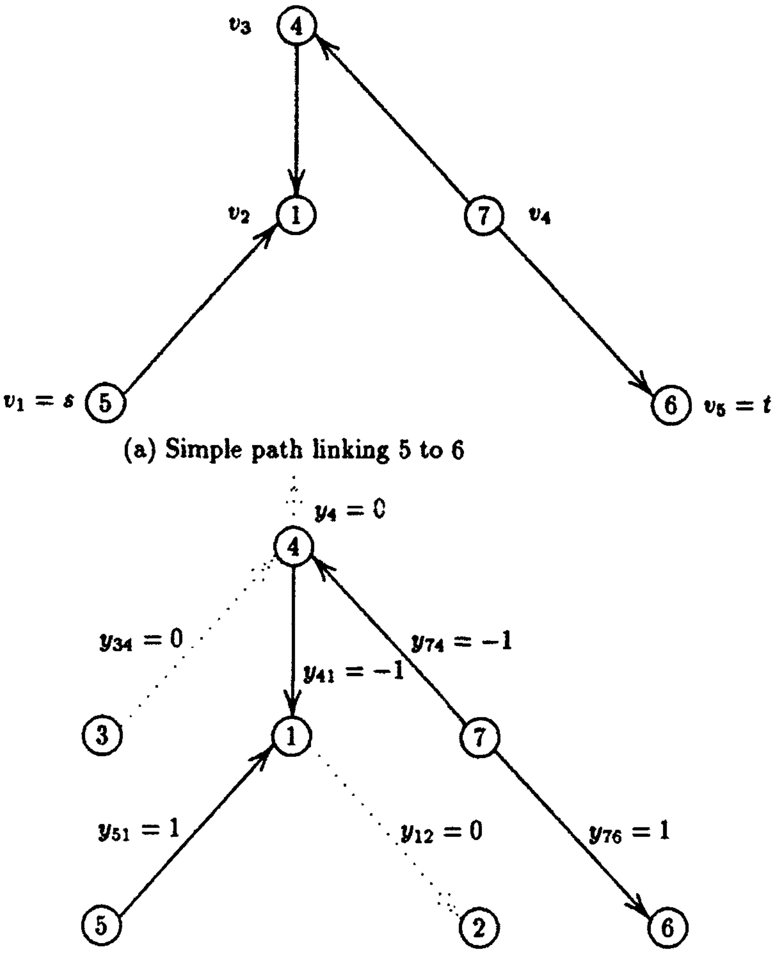


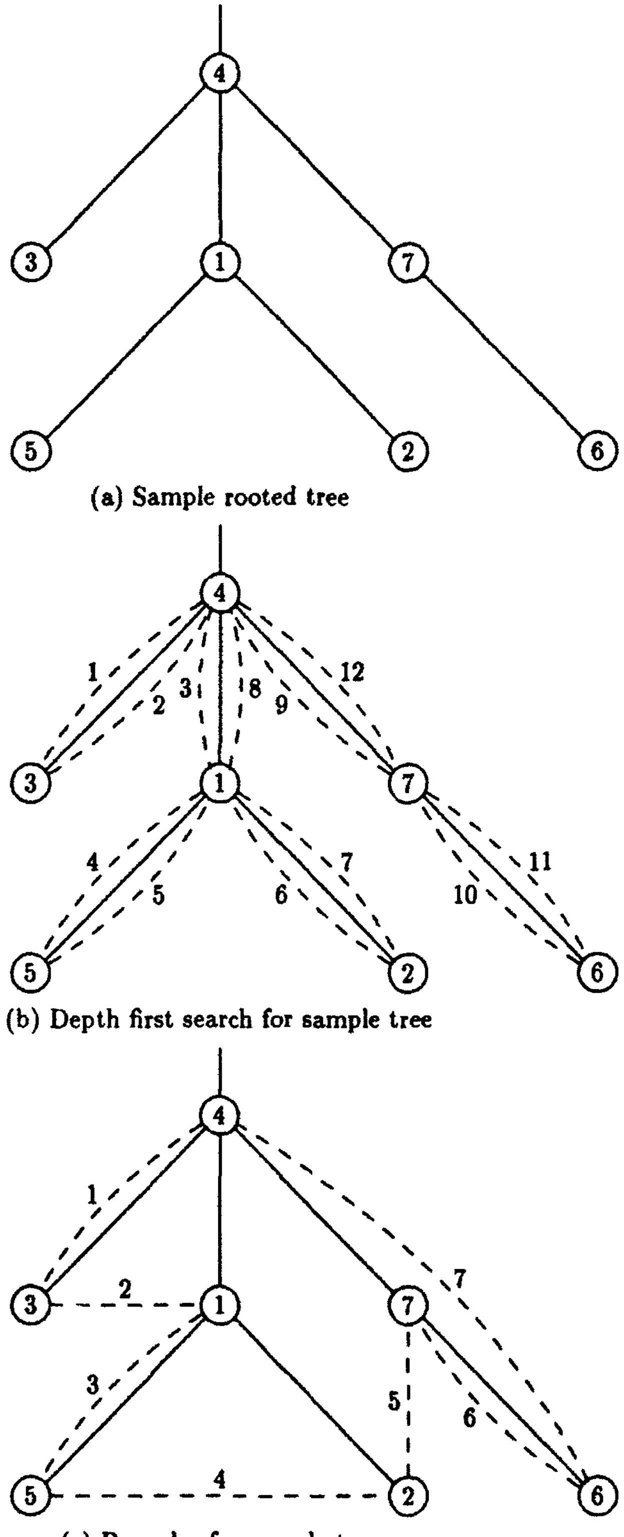
Figure 9: A simple path linking s to t



(a) Simple path linking 5 to 6

(b) Updated column for e_{56}

Figure 10: Updated column example



(a) Sample rooted tree

(b) Depth first search for sample tree

(c) Preorder for sample tree

Figure 11: Illustration of thread labels for sample root:d tree

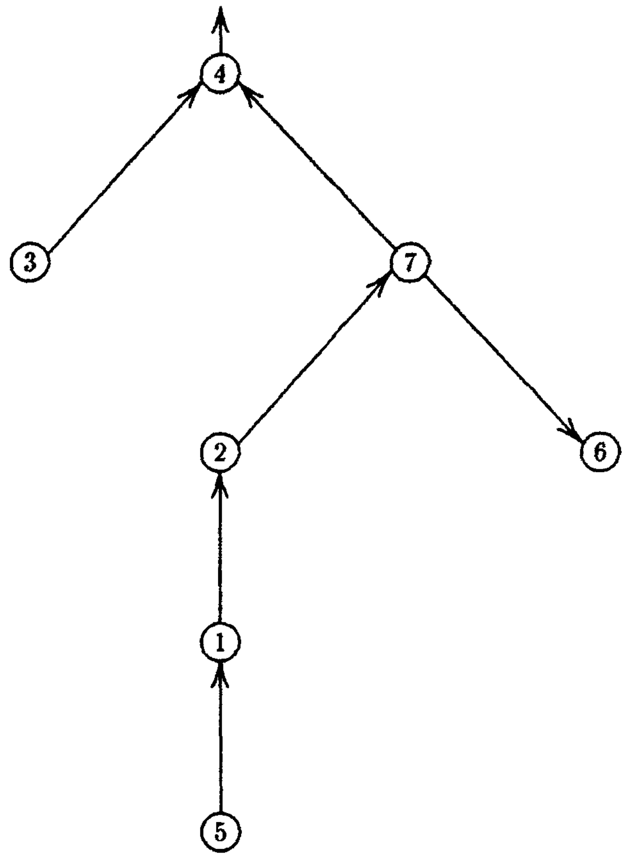


Figure 12: New rooted tree after basis exchange update

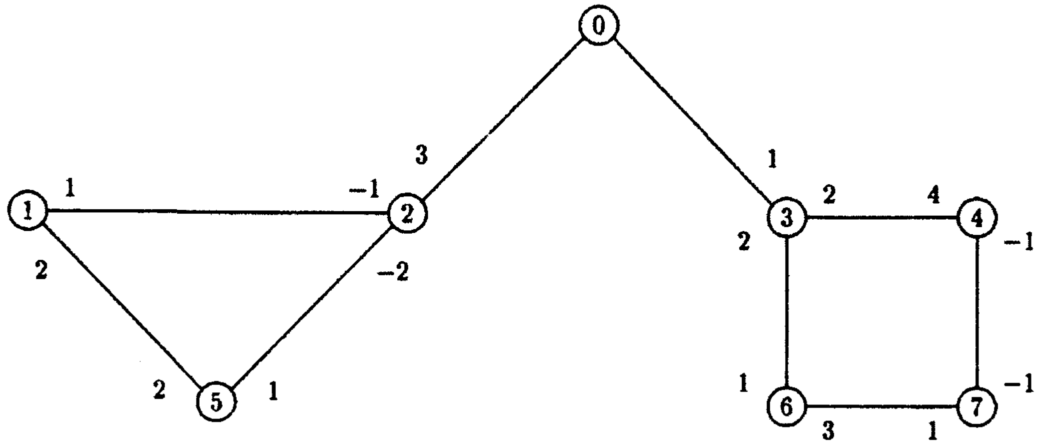


Figure 13: Example generalized network

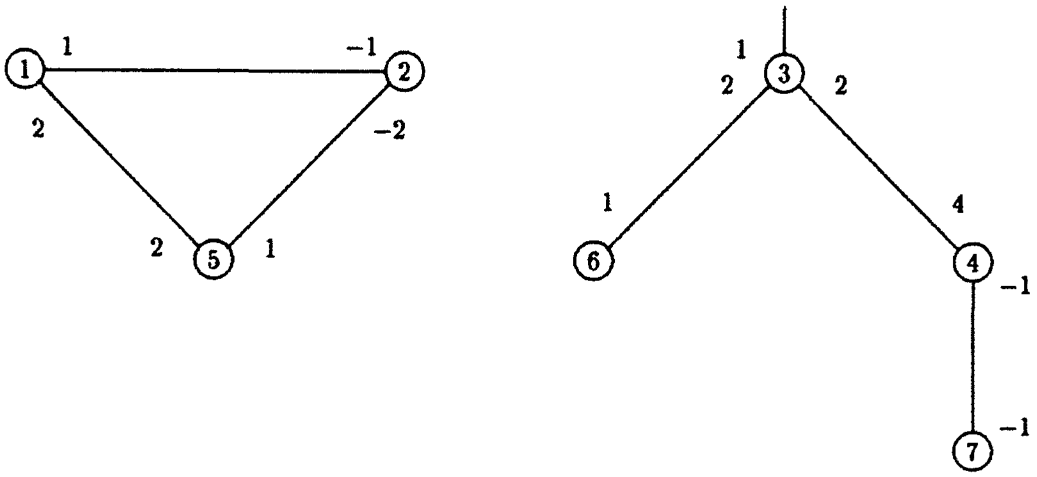


Figure 14: Basis for a generalized network

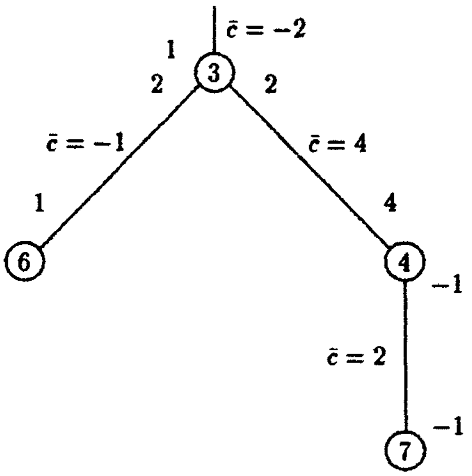


Figure 15: Rooted tree component of a basis for a generalized network

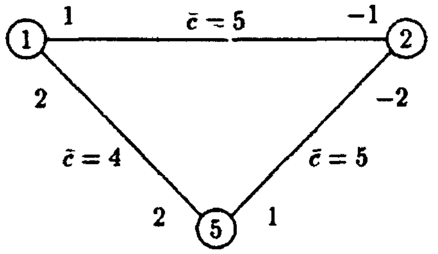


Figure 16: One-tree component of a basis for a generalized network

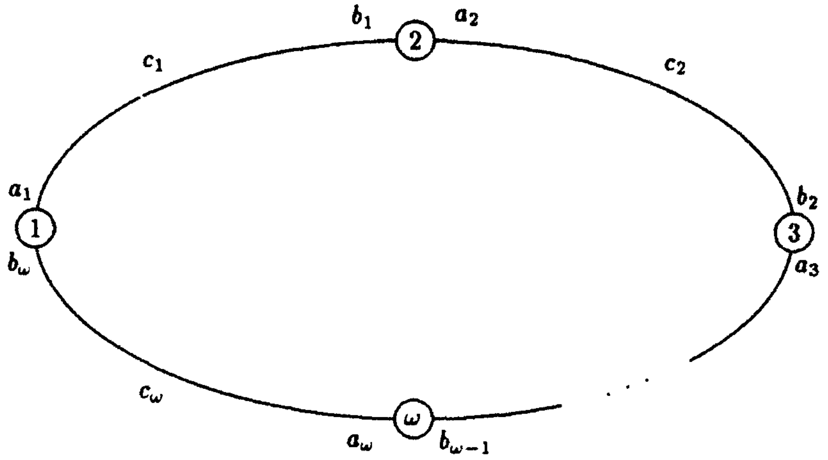


Figure 17: Cycle for a generalized network

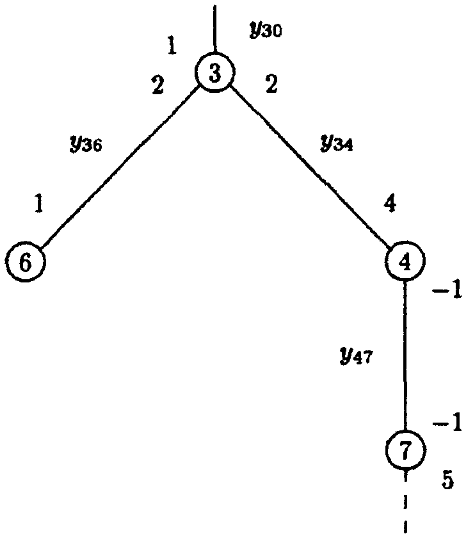


Figure 18: Column update for a rooted tree component of a basis for a generalized network

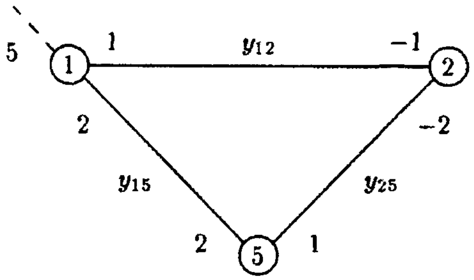


Figure 19: Column update for a one-tree component of a basis for a generalized network

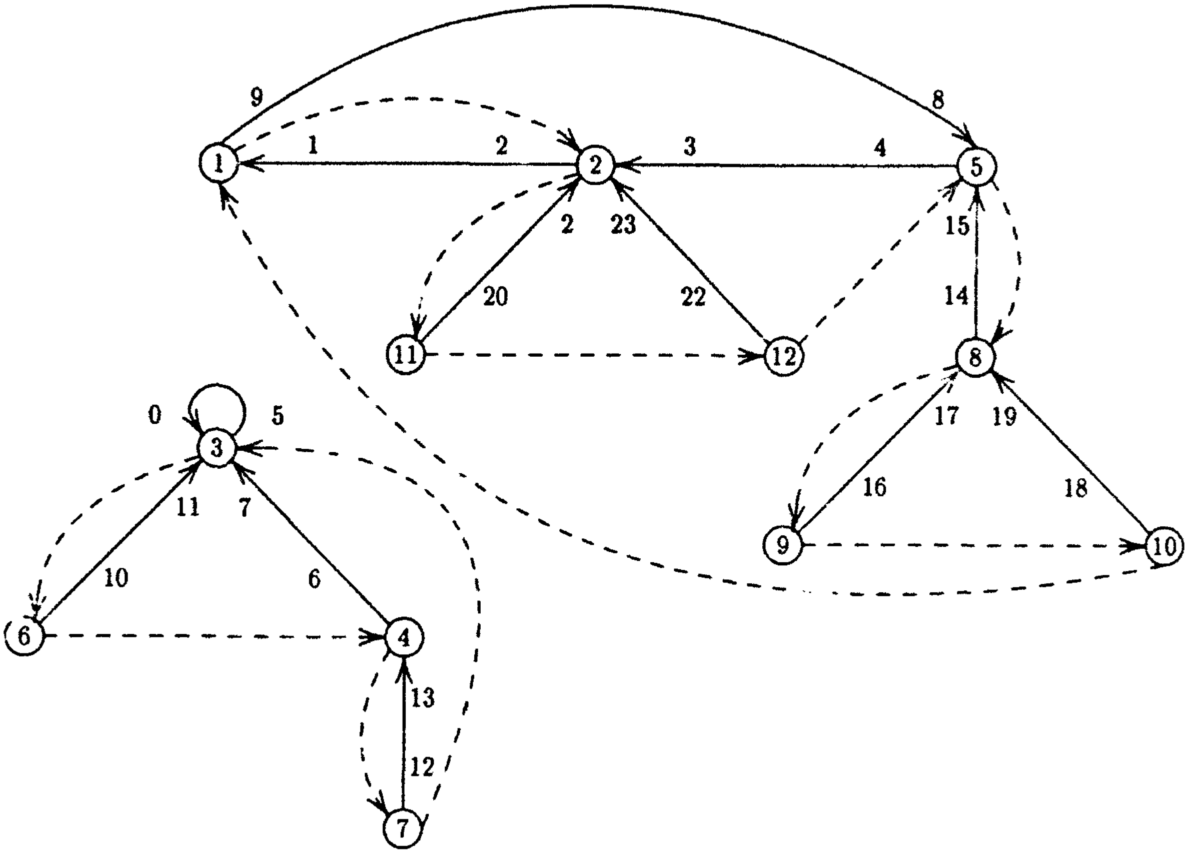


Figure 20: Basis for a generalized network with the predecessor and thread labels illustrated

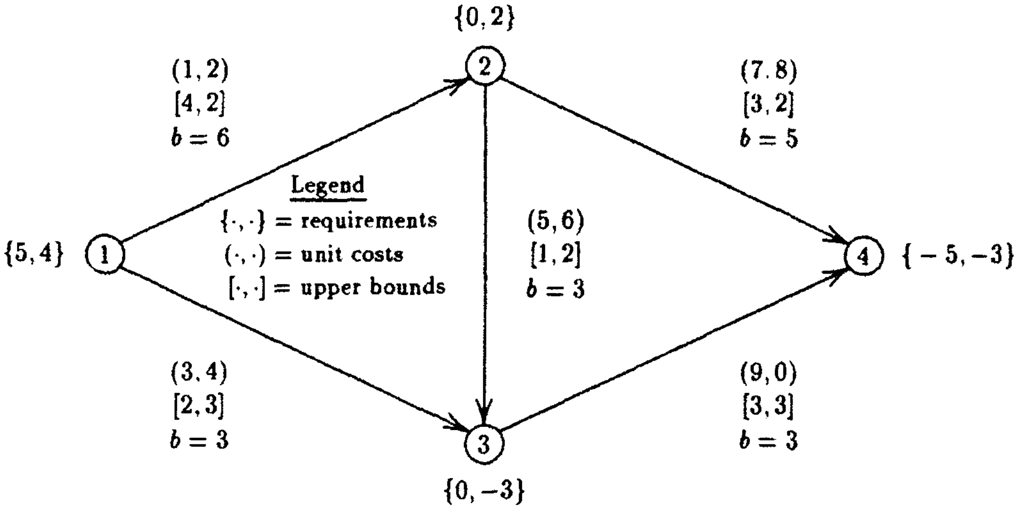


Figure 21: Example two-commodity multicommodity network

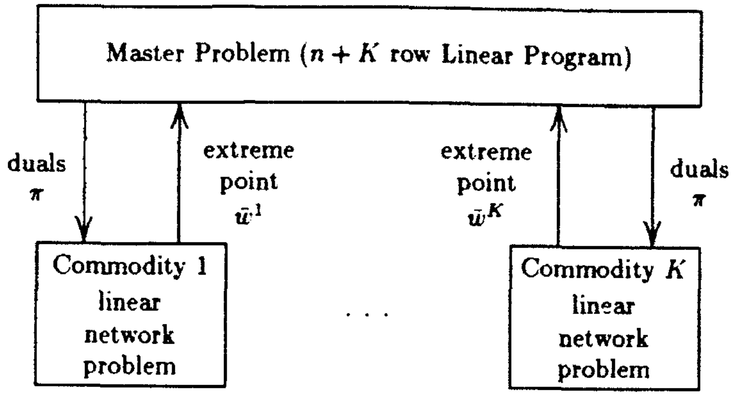

Master Problem ($n + K$ row Linear Program)
duals π
extreme point $\bar{w}^1$
extreme point $\bar{w}^K$
duals π
Commodity 1 linear network problem
. . .
Commodity K linear network problem

Figure 22: The price-directive decomposition algorithm

APPENDIX B

The One-to-One Shortest-Path Problem: An Empirical Analysis With the Two-Tree Dijkstra Algorithm

Richard V. Helgason †
Jeffery L. Kennington †
B. Douglas Stewart ‡

Four new shortest-path algorithms, two sequential and two parallel, for the source to sink shortest-path problem are presented and empirically compared with five algorithms previously discussed in the literature. The new algorithm, S22, combines the highly effective data structure of the S2 algorithm of Dial, Glover, Karney, and Klingman, with the idea of simultaneously building shortest-path trees from both source and sink nodes, and was found to be the fastest sequential shortest-path algorithm. The new parallel algorithm, PS22, is based on S22 and is the best of the parallel algorithms. We also present results for three new S22-type shortest-path heuristics. These heuristics find very good (often optimal) paths much faster than the best shortest-path algorithm.

Revised (December 1992)

Since the late fifties when its first solution methods were developed, the shortest-path problem has become one of the fundamental problems in the areas of combinatorial optimization, computer science, and operations research. Algorithms and applications are commonly found in the important books in these areas (see for example Berge and Ghouila-Houri (1962), Bertsekas and Gallager (1987), Even (1979), Hu (1982), Jensen and Barnes (1980), Lawler (1987), Papadimitriou and Steiglitz (1987), Quinn (1984), Rockafellar (1984), and Tarjan (1983)). The study of this problem has been motivated by both its elegant mathematical structure and its many practical applications. Our recent interest in this problem was occasioned by the need to solve shortest-path subproblems in several mathematical optimization procedures we are developing in an MIMD parallel computing environment.

Excellent surveys of the many shortest-path problem variations may be found in Deo and Pang (1984) and Gallo and Pallottino (1986). A survey of techniques and computational comparisons may be found in Gallo and Pallottino (1988), in Dial, Glover, Karney, and Klingman (1977) and (1979), in Klingman, Mote, and Whitman (1978), in Glover, Glover, and Klingman (1984), in Desrochers (1987), and in Divoky (1987). The methods are grouped into two general classes: *label-setting algorithms* and *label-correcting algorithms*. Dijkstra (1959) is credited with the first label-setting algorithm and any algorithm that uses this approach has been considered a particular implementation of Dijkstra's original algorithm (see Gallo and Pallottino (1986)).

Typically, the shortest-path problem is one that requires the shortest-path from a single source node, say s, to all other nodes in a network. The solution can be represented as a shortest-path tree rooted at s. In this paper we are concerned with the problem of finding the shortest-path between a source node and a sink node, t. Dantzig (1960) suggested that a pair of trees be built with one rooted at s and the other rooted at t. No stopping criteria were given. This strategy also appears in the book by Berge and Ghouila-Houri (1962) with an incorrect stopping criterion. Nicholson (1966) was the first to present a correct analysis of the Dijkstra

† Department of Computer Science and Engineering, Southern Methodist University, Dallas, TX 75275-0122.

‡ Department of industrial Engineering, University of Alabama, Tuscaloosa, AL 35406-0288.

This research was supported in part by the Department of Defense under Contract Number MDA 903-86-C0182, the Air Force Office of Scientific Research under Contract Number AFOSR F49620-92-J-0032, and the Office of Naval Research under Contract Number N00014-87-K-0223.

two-tree algorithm. Hart, Nilsson, and Raphael (1968) presented a one-tree algorithm utilizing heuristic cost functions. They included the case in which lower bounds on distances between nodes are available and proved that optimal paths are obtained. Pohl (1971) extended these results for the bi-directional algorithm, antedating Mohr and Pasche (1988), who presented similar results. Additional discussion may be found in the survey by Dreyfus (1969), where he conjectured that building trees from s and t would be ineffective for this problem.

Few empirical studies have been reported in the literature for the two-tree algorithms. Pohl (1969) presented results that have received little recognition, and no further studies appear until recently. Mohr and Pasche (1988) solved for shortest-paths in three grid networks and a network representing a road map of Switzerland, using the two-tree Dijkstra algorithm as well as their version of the two-tree algorithm for networks with lower bounds available. They also simulated results for parallel algorithms if two processors were available.

The motivation for this study was to implement two-tree algorithms using more efficient data structures than the classical Dijkstra algorithm, and to actually solve shortest-path problems using two processors. Four new algorithms were developed: sequential and parallel two-tree algorithms based on Dial (1969) (the S1 code in Dial et al. (1979)), and sequential and parallel two-tree algorithms based on the S2 code in Dial et al. (1979). These codes are compared with the classical Dijkstra, S1, S2, two-tree Dijkstra, and parallel two-tree Dijkstra codes.

While performing this study, it was noted that the two-tree S2 algorithm finds "good" paths quickly. Three heuristic path algorithms were developed by simply stopping early while executing the two-tree S2 algorithm. Often times these heuristics find a shortest-pat'., although they do not prove it is so, and these paths are found much faster than the fastest optimal algorithm. We present computation times as well as measures of closeness to optimality for these heuristics.

1. THE ALGORITHMS

This section presents the definitions, notation, and nine algorithms representing the codes used in our computational study. The notation and presentation are based on that found in Gallo and Pallattino (1986). The input for each algorithm is a directed graph $G = [N,A]$ with a node set N and an arc set A. Associated with each arc $(i,j) \in A$ is a length l_{ij}. A shortest-path is desired between two nodes s and t and it is assumed that such a path exists. We also assume for all algorithms that $l_{ij} \geq 0$ for all $(i,j) \in A$. For efficient implementation it is assumed that G is given in the form of arc-lists. Specifically, the forward star for node $u \in N$, $FS(u)$, is the set of all arcs $(u,j) \in A$. Six algorithms require a backward star, $BS(v)$, defined as the set of all arcs $(i,v) \in A$ for node $v \in N$.

The basic working entities of each algorithm include a set of labels, d_u, for node distances from the root of a shortest-path tree T, a set of predecessors, p_u, for nodes in the tree, and the set of candidate nodes, Q. In the algorithms based on the S1 and S2 algorithms, the set Q will be divided into subsets, or buckets, that will contain candidate nodes that are the same distance from the root node. This requires that each arc length be *integer* to correspond to an index. For ease of presentation, we will let Γ be the set of indices, $\{0, ..., lmax\}$ for the buckets of Q, where $lmax = \max\{l_{uv} \cdot (u,v) \in A\}$. The notation will be modified for the bi-directional algorithms by using the superscripts s or t to indicate the root of the tree. The algorithms terminate with the length of the shortest-path from s to t and a small set of nodes, J, used to identify a shortest-path. A shortest-path from s to t is implicit in the predecessor labels.

1.1 The classical Dijkstra algorithm

Dijkstra's classical algorithm (1959) begins at node s and builds a shortest-path tree T in which the shortest-path from s to any node in the tree is known. When node t is placed in the tree we have a minimum length directed path from s to t and may terminate. As mentioned before, the algorithms discussed here differ in the way the candidate nodes are placed in and retrieved from the set Q. The implementation here for the classical Dijkstra algorithm (D1) searches the list of candidate nodes, Q, for minimum label nodes and places all such nodes in the set R. Then all the nodes in R can be scanned one after the other. When there are many nodes tied with the minimum label, searches are avoided with only minimal effort. The algorithm may be stated as follows:

```
Procedure D1($s,t$)
   begin
      initialize:
         $p_i \leftarrow 0, d_i \leftarrow \infty$ for all $i \in N$; $Q \leftarrow \emptyset$;
         $d_s \leftarrow 0, p_s \leftarrow s, R \leftarrow \{s\}, T \leftarrow \emptyset$;
      while $t \notin R$
         for each $u \in R$ do
            for each $(u,v) \in FS(u)$ such that $d_u + l_{uv} < d_v$ do
               $d_v \leftarrow d_u + l_{uv}$;
               $p_v \leftarrow u$;
               if $v \notin Q$ then $Q \leftarrow Q \cup \{v\}$;
            end
            $T \leftarrow T \cup \{u\}$;
         end
         comment: search $Q$ for minimum label nodes and place in $R$
         $\alpha \leftarrow \min\{d_i : i \in Q\}, R \leftarrow \{i : d_i = \alpha\}, Q \leftarrow Q \backslash R$;
      endwhile
   end
```

1.2 The S1 algorithm

This one-tree algorithm is implemented as proposed by Dial (1969). As in algorithm D1, it selects a minimum label node to scan each iteration, but the nodes in Q are stored in buckets according to their distance labels. More specifically, $lmax + 1$ buckets are required, where $lmax = \max\{l_{uv} : (u,v) \in A\}$ and a node u is stored in bucket z if $d_u = z(\text{mod } lmax + 1)$. Only nodes with equal distance labels will be in bucket z and only $lmax + 1$ buckets are required, because for a node u with minimum label, we have that for each $v \in Q, d_u \leq d_v \leq d_u + lmax$. The buckets are implemented efficiently as two-way linked-lists. From minimum label node u, the next non-empty bucket contains the next minimum label node(s) and the effort to search an entire list as in algorithm D1 is greatly reduced. The trade-off is increased effort managing the two-way linked-list each time a node has an improved distance label. The algorithm may be stated as follows:

```
Procedure S1($s, t$)
  begin
    initialize:
      $p_i \leftarrow 0, d_i \leftarrow \infty$ for all $i \in N$; $Q_z \leftarrow \emptyset$ for $z = 1, ..., lmax$;
      $Q_0 \leftarrow \{s\}$, $d_s \leftarrow 0, p_s \leftarrow s$, $T \leftarrow \emptyset$;
    while $t \notin T$
      let $z$ be the next index such that $Q_z \neq \emptyset$ ;
      for each $u \in Q_z$ do
        $Q_z \leftarrow Q_z \backslash \{u\}$;
        for each $(u, v) \in FS(u)$ such that $d_u + l_{uv} < d_v$ do
          $a \leftarrow d_v (\text{mod } lmax + 1)$;
          $d_v \leftarrow d_u + l_{uv}$;
          $b \leftarrow d_v (\text{mod } lmax + 1)$;
          $p_v \leftarrow u$;
          $Q_a \leftarrow Q_a \backslash \{v\}$;
          $Q_b \leftarrow Q_b \cup \{v\}$;
        end
        $T \leftarrow T \cup \{u\}$;
      end
    endwhile
  end
```

1.3 The S2 algorithm

This one-tree algorithm is based on an idea due to Dantzig (1960) and the implementation here is due to Dial et al. (1977). It requires that each $FS(u)$ for all $u \in N$ be sorted in shortest first order. Given this, the observation can be made that the entire forward star of a node u need not be scanned all at once in that the node, say v, that is first updated will have a distance label less than or equal to any subsequent nodes updated from node u. The node v is placed on a one-way linked-list, paired with u, at a level $d_u + l_{uv}$. In stating the algorithm below, we use $k(u)$ as a counter to point to the next arc in $FS(u)$ to be scanned. The node $w_{k(u)}$ is the corresponding node of this arc. The length of each forward arc-list is given by $h(u)$. It should be noted that the actual implementation does not use a counter. A simple minus sign in the forward star array can indicate the next arc to be scanned.

Nodes may be duplicated on the linked-list, therefore no deleting is required. Even so, the number of nodes on the linked-list will not exceed the total number of nodes since only one per scanned node is allowed. Q is searched as in algorithm S1 for the next non-empty bucket and minimum label node. If the node's paired predecessor is not its current predecessor, the node is already in the shortest-path tree and may be discarded. The algorithm may be stated as follows (see Gallo and Pallottino (1986)):

```
Procedure S2($s, t$)
  begin
    initialize:
      $p_i \leftarrow 0, d_i \leftarrow \infty, k(i) \leftarrow 1, h(i) = |FS(i)|$ for all $i \in N$;
      $Q_z \leftarrow \emptyset$ for $z = 1, ..., lmax$; $Q_0 \leftarrow \{s\}$, $d_s \leftarrow 0, p_s \leftarrow s$, $T \leftarrow \emptyset$;
    while $t \notin T$
      let $z$ be the next index such that $Q_z \neq \emptyset$;
      for each $u \in Q_z$ do
        comment: determine next node in $FS(u)$
        $v \leftarrow w_{k(u)}$;

        $Q_z \leftarrow Q_z \backslash \{u\}$;
        comment: determine new candidate arc
        INSERT($u$);
        $T \leftarrow T \cup \{u\}$;
        if $v \notin Q$ then INSERT($v$);
      end
    endwhile
  end
Procedure INSERT($x$)
  begin
    $k(x) \leftarrow k(x) + 1$;
    $y \leftarrow w_{k(x)}$;

    while $k(x) \leq h(x)$ and $d_x + l_{xy} \geq d_y$ do
      $k(x) \leftarrow k(x) + 1$;
      $y \leftarrow w_{k(x)}$;
    endwhile
    if $k(x) \leq h(x)$ then
      $p_y \leftarrow x$;
      $d_y \leftarrow d_x + l_{xy}$;
      $a \leftarrow d_y(\text{mod } lmax + 1)$;
      $Q_a \leftarrow Q_a \cup \{x\}$;
    endif
  end
```

1.4 The two-tree Dijkstra algorithm

The two-tree Dijkstra algorithm builds a pair of shortest-path trees, one from s and one from t. The tree rooted at t is analogous to the one rooted at s, but scans backward stars, and its predecessors are the heads of arcs rather than tails. The two trees are grown in alternate steps and termination is triggered when a node appears in both trees. It should be noted however that the node appearing in both trees is not necessarily on a shortest-path (see Nicholson (1966) for a proof of termination criteria). Testing on random graphs showed that about 72% of the time this node will be on the shortest-path. A search is performed over nodes in both trees to find a node, say r, such that $d^s_r + d^t_r$ is a minimum. A shortest-path from s to t may be found by following predecessors from r to s and from r to t in each tree, respectively. We define J as the set of nodes which can be used to identify a shortest-path. The algorithm requires twice the storage of algorithm D1 and may be stated as follows:

```
Procedure D2($s, t$)
  begin
    initialize:
      $p_i^s \leftarrow 0, d_i^s \leftarrow \infty$ for all $i \in N$; $Q^s \leftarrow \emptyset$;
      $d_s^s \leftarrow 0, p_s^s \leftarrow s, T^s \leftarrow \emptyset, R^s \leftarrow \{s\}$;
      $p_i^t \leftarrow 0, d_i^t \leftarrow \infty$ for all $i \in N$; $Q^t \leftarrow \emptyset$;
      $d_t^t \leftarrow 0, p_t^t \leftarrow t, T^t \leftarrow \emptyset, R^t \leftarrow \{t\}$;
    while $T^s \cap T^t = \emptyset$ do
      for each $u \in R^s$ do
        for each $(u, v) \in FS(u)$ such that $d_u^s + l_{uv} < d_v^s$ do
          $d_v^s \leftarrow d_u^s + l_{uv}$;
          $p_v^s \leftarrow u$;
          if $v \notin Q^s$ then $Q^s \leftarrow Q^s \cup \{v\}$;
        end
        $T^s \leftarrow T^s \cup \{u\}$;
      end
      comment: search $Q^s$ for minimum label nodes and place in $R^s$
      $\alpha \leftarrow \min\{d_i^s : i \in Q^s\}, R^s \leftarrow \{i : d_i^s = \alpha\}, Q^s \leftarrow Q^s \backslash R^s$;
      for each $v \in R^t$ do
        for each $(u, v) \in BS(v)$ such that $d_v^t + l_{uv} < d_u^t$ do
          $d_u^t \leftarrow d_v^t + l_{uv}$;
          $p_u^t \leftarrow v$;
          if $v \notin Q^t$ then $Q^t \leftarrow Q^t \cup \{u\}$;
        end
        $T^t \leftarrow T^t \cup \{v\}$;
      end
      comment: search $Q^t$ for minimum label nodes and place in $R^t$
      $\alpha \leftarrow \min\{d_i^t : i \in Q^t\}, R^t \leftarrow \{i : d_i^t = \alpha\}, Q^t \leftarrow Q^t \backslash R^t$;
    endwhile
    comment: stopping criterion met
    $\beta \leftarrow \min\{d_i^s + d_i^t : i \in T^s \cup T^t\}$;
    $J \leftarrow \{i \in T^s \cup T^t : d_i^s + d_i^t = \beta\}$;
  end
```

1.5 The two-tree S1 algorithm

The two-tree S1 algorithm builds a pair of shortest-path trees, one from s and one from t, using S1 data structures for each tree. As in algorithm D2, termination is triggered when a node is first found to be in both trees. The node r such that $d_r^s + d_r^t$ is minimum gives the shortest-distance from s to t and lies on a shortest-path. The algorithm requires twice the storage of S1 and may be stated as follows:

```
Procedure S12($s, t$)
  begin
    initialize:
      $p_i^s \leftarrow 0, d_i^s \leftarrow \infty$ for all $i \in N$; $Q_z^s \leftarrow \emptyset$ for $z = 1, ..., lmax$;
      $Q_0^s \leftarrow \{s\}$, $d_s^s \leftarrow 0, p_s^s \leftarrow s$, $T^s \leftarrow \emptyset$;
      $p_i^t \leftarrow 0, d_i^t \leftarrow \infty$ for all $i \in N$; $Q_z^t \leftarrow \emptyset$ for $z = 1, ..., lmax$;
      $Q_0^t \leftarrow \{t\}$, $d_t^t \leftarrow 0, p_t^t \leftarrow t$, $T^t \leftarrow \emptyset$;
    while $T^s \cap T^t = \emptyset$ do
      let $z$ be the next index such that $Q^s_z \neq \emptyset$ ;
      for each $u \in Q^s_z$ do
        $Q^s_z \leftarrow Q^s_z \backslash \{u\}$;
        for each $(u,v) \in FS(u)$ such that $d^s_u + l_{uv} < d^s_v$ do
          $a \leftarrow d^s_v (\text{mod } lmax + 1)$;
          $d^s_v \leftarrow d^s_u + l_{uv}$;
          $b \leftarrow d^s_v (\text{mod } lmax + 1)$;
          $p^s_v \leftarrow u$;
          $Q^s_a \leftarrow Q^s_a \backslash \{v\}$;
          $Q^s_b \leftarrow Q^s_b \cup \{v\}$;
        end
        $T^s \leftarrow T^s \cup \{u\}$;
      end
      let $z$ be the next index such that $Q^t_z \neq \emptyset$ ;
      for each $v \in Q^t_z$ do
        $Q^t_z \leftarrow Q^t_z \backslash \{v\}$;
        for each $(u,v) \in BS(v)$ such that $d^t_v + l_{uv} < d^t_u$ do
          $a \leftarrow d^t_u (\text{mod } lmax + 1)$;
          $d^t_u \leftarrow d^t_v + l_{uv}$;
          $b \leftarrow d^t_u (\text{mod } lmax + 1)$;
          $p^t_u \leftarrow v$;
          $Q^t_a \leftarrow Q^t_a \backslash \{u\}$;
          $Q^t_b \leftarrow Q^t_b \cup \{u\}$;
        end
        $T^t \leftarrow T^t \cup \{v\}$;
      end
    endwhile
    comment: stopping criterion met
    $\beta \leftarrow \min\{d^s_i + d^t_i : i \in T^s \cup T^t\}$;
    $J \leftarrow \{i \in T^s \cup T^t : d^s_i + d^t_i = \beta\}$;
  end
```

1.6 The two-tree S2 algorithm

As in the previous two-tree algorithms, the two-tree S2 algorithm uses mirror S2 data structures to build two shortest-path trees. At first one would expect the stopping procedure to be the same as in the previous two-tree algorithms, namely, when a node is in both trees, find the minimum $d^s_i + d^t_i$ for all $i \in T^s \cup T^t$. However, at the time a node is first placed in its second tree, we are not quite ready to search for such a minimum doubly labelled node. In Nicholson (1966) such a node is proven to be on a shortest-path because each node in both trees has had its arc-list fully scanned. In the S2 implementation for each tree this is not the case. In fact, this is the advantage of the one-tree S2 algorithm. In two directions we must perform additional scanning to meet the Nicholson criterion, however, we will not need to manage the linked-lists. All that is needed is to update distance labels and predecessors. Actually we need only scan a subset of arcs from each arc-list. Nicholson proves that any node that is not in either tree when a node is first in both trees will not be on a shortest-path. If arcs were scanned to these nodes, they would still not be in either tree or on a shortest-path, so updating their distance labels would be wasted effort. These nodes also make up the majority of the arc-lists. The only arcs that need to be scanned during the "mop-up" phase are those arcs that have from nodes in one tree and to nodes in the other tree. Since these arcs may be scanned from either node, it is more efficient to consider these arcs from the nodes in the smaller tree. After updating distance labels and predecessors we are ready to search for the minimum doubly labelled node to find our minimum distance from s to t and a shortest-path. The algorithm may be stated as follows:

```
Procedure S22($s, t$)
  begin
    initialize:
      $p_i^s \leftarrow 0, d_i^s \leftarrow \infty, fk(i) \leftarrow 1, fh(i) = |FS(i)|$ for all $i \in N$;
      $Q_z^s \leftarrow \emptyset$ for $z = 1, ..., lmax$; $Q_0^s \leftarrow \{s\}$, $d_s^s \leftarrow 0, p_s^s \leftarrow s$, $T^s \leftarrow \emptyset$;
      $p_i^t \leftarrow 0, d_i^t \leftarrow \infty, bk(i) \leftarrow 1, bh(i) = |BS(i)|$ for all $i \in N$;
      $Q_z^t \leftarrow \emptyset$ for $z = 1, ..., lmax$; $Q_0^t \leftarrow \{t\}$, $d_t^t \leftarrow 0, p_t^t \leftarrow t$, $T^t \leftarrow \emptyset$;
    while $T^s \cap T^t = \emptyset$ do
      let $z$ be the next index such that $Q_z^s \neq \emptyset$ ;
      for each $u \in Q_z^s$ do
        comment: determine next node in $FS(u)$
        $v \leftarrow w_{fk(u)}$;

        $Q_z^s \leftarrow Q_z^s \backslash \{u\}$;
        comment: determine new candidate arc
        SINSERT($u$);
        $T^s \leftarrow T^s \cup \{u\}$;
        if $v \notin Q^s$ then SINSERT($v$);
      end
      let $z$ be the next index such that $Q_z^t \neq \emptyset$ ;
      for each $v \in Q_z^t$ do
        comment: determine next node in $BS(u)$
        $u \leftarrow w_{bk(v)}$;

        $Q_z^t \leftarrow Q_z^t \backslash \{v\}$;
        comment: determine new candidate arc
        TINSERT($v$);
        $T^t \leftarrow T^t \cup \{v\}$;
        if $u \notin Q^t$ then TINSERT($u$);
      end
    endwhile
    comment: mop-up phase from smaller tree
    if $|T^s| < |T^t|$ then
      for each $u \in T^s$ do
        for each $(u, y) \in FS(u)$ do
          if $y \in T^t$ then
            if $d_u^s + l_{uy} < d_y^s$ then
              $d_y^s \leftarrow d_u^s + l_{uy}$;
              $p_y^s \leftarrow u$;
            endif
          endif
        end
      end
    else
      for each $v \in T^t$ do
        for each $(w, v) \in BS(v)$ do
          if $w \in T^s$ then
            if $d_v^t + l_{wv} < d_w^t$ then
              $d_w^t \leftarrow d_v^t + l_{wv}$;
              $p_w^t \leftarrow v$;
            endif
          endif
        end
      end
    endif
```

```
      comment: stopping criterion met
      $\beta \leftarrow \min\{d_i^s + d_i^t : i \in T^s \cup T^t\}$;
      $J \leftarrow \{i \in T^s \cup T^t : d_i^s + d_i^t = \beta\}$;
   end
Procedure SINSERT($x$)
   begin
      $fk(x) \leftarrow fk(x) + 1$;
      $y \leftarrow w_{fk(x)}$;
      while $fk(x) \le fh(x)$ and $d_x^s + l_{xy} \ge d_y^s$ do
         $fk(x) \leftarrow fk(x) + 1$;
         $y \leftarrow w_{fk(x)}$;
      endwhile
      if $fk(x) \le fh(x)$ then
         $p_y^s \leftarrow x$;
         $d_y^s \leftarrow d_x^s + l_{xy}$;
         $a \leftarrow d_y^s(\mathrm{mod}\ lmax + 1)$;
         $Q_a^s \leftarrow Q_a^s \cup \{x\}$;
      endif
   end
Procedure TINSERT($x$)
   begin
      $bk(x) \leftarrow bk(x) + 1$;
      $y \leftarrow w_{bk(x)}$;
      while $bk(x) \le bh(x)$ and $d_x^t + l_{yx} \ge d_y^t$ do
         $bk(x) \leftarrow bk(x) + 1$;
         $y \leftarrow w_{bk(x)}$;
      endwhile
      if $bk(x) \le bh(x)$ then
         $p_y^t \leftarrow x$;
         $d_y^t \leftarrow d_x^t + l_{yx}$;
         $a \leftarrow d_y^t(\mathrm{mod}\ lmax + 1)$;
         $Q_a^t \leftarrow Q_a^t \cup \{x\}$;
      endif
   end
```

1.7 The parallel two-tree Dijkstra algorithm

It is readily observed that in the two-tree shortest-path algorithms, the trees are independent of each other. The only requirement is to check whether or not a node is in the opposite tree. This read-only step causes no interference using multiple processors. This leads to the simplest asynchronous parallel application using two processors, one for each tree. When one processor recognizes that a node is in both trees, it sets a flag to tell the other processor to find the minimum doubly labelled node in its tree while it does the same. The minimum of the two is the minimum path distance from s to t. Again, a shortest-path is implicit in the predecessor labels beginning with the minimum doubly labelled node. In the algorithms below, each processor has its own indentifying number called *procid*. The parallel processing construct called *fork(2)* indicates that two processors are to be used to execute the sections until the *join* construct is reached. The algorithm may be stated as follows:

```
Procedure PD2($s, t$)
   begin
      $flag \leftarrow 0$;
      comment: begin use of two processors
      fork(2)
      if $procid = 1$ then
         initialize:
            $p_i^s \leftarrow 0, d_i^s \leftarrow \infty$ for all $i \in N$; $Q^s \leftarrow \emptyset$;
            $d_s^s \leftarrow 0, p_s^s \leftarrow s, T^s \leftarrow \emptyset, R^s \leftarrow \{s\}$;
         synchronize processors
         while $flag = 0$ do
            for each $u \in R^s$ do
               for each $(u, v) \in FS(u)$ such that $d_u^s + l_{uv} < d_v^s$ do
                  $d_v^s \leftarrow d_u^s + l_{uv}$;
                  $p_v^s \leftarrow u$;
                  if $v \notin Q^s$ then $Q^s \leftarrow Q^s \cup \{v\}$;
               end
               $T^s \leftarrow T^s \cup \{u\}$;
               if $u \in T^t$ then $flag \leftarrow 1$;
            end
            comment: search $Q^s$ for minimum label nodes and place in $R^s$
            $\alpha \leftarrow \min\{d_i^s : i \in Q^s\}, R^s \leftarrow \{i : d_i^s = \alpha\}, Q^s \leftarrow Q^s \backslash R^s$;
         endwhile
         $\beta_s \leftarrow \min\{d_i^s + d_i^t : i \in T^s\}$;
      endif
      if $procid = 2$ then
         initialize:
            $p_i^t \leftarrow 0, d_i^t \leftarrow \infty$ for all $i \in N$; $Q^t \leftarrow \emptyset$;
            $d_t^t \leftarrow 0, p_t^t \leftarrow t, T^t \leftarrow \emptyset, R^t \leftarrow \{t\}$;
         synchronize processors
         while $flag = 0$ do
            for each $v \in R^t$ do
               for each $(u, v) \in BS(v)$ such that $d_v^t + l_{uv} < d_u^t$ do
                  $d_u^t \leftarrow d_v^t + l_{uv}$;
                  $p_u^t \leftarrow v$;
                  if $v \in Q^t$ then $Q^t \leftarrow Q^t \cup \{u\}$;
               end
               $T^t \leftarrow T^t \cup \{v\}$;
               if $v \in T^s$ then $flag \leftarrow 1$;
            end
            comment: search $Q^t$ for minimum label nodes and place in $R^t$
            $\alpha \leftarrow \min\{d_i^t : i \in Q^t\}, R^t \leftarrow \{i : d_i^t = \alpha\}, Q^t \leftarrow Q^t \backslash R^t$;
         endwhile
         $\beta_t \leftarrow \min\{d_i^s + d_i^t : i \in T^t\}$;
      endif
      comment: end use of multiple processors
      join processors
      $\beta \leftarrow \min\{\beta_s, \beta_t\}$;
      $J \leftarrow \{i \in T^s \cup T^t : d_i^s + d_i^t = \beta\}$;
   end
```

1.8 The parallel two-tree S1 algorithm

The parallel two-tree S1 algorithm is similar to the parallel two-tree Dijkstra algorithm. Each processor is assigned one of the nodes, s or t, and builds a tree using the S1 data structure. When a node is found to be in both trees, a flag is set to tell both processors to find the minimum doubly labelied noue in its respective tree. The minimum of these two gives the minimum distance path from s to t with a path implicit in the predecessor labels. The algorithm may be stated as follows:

```
Procedure PS12($s, t$)
  begin
    $flag \leftarrow 0$;
    comment: begin use of two processors
    fork(2)
    if $procid = 1$ then
      initialize:
        $p_i^s \leftarrow 0, d_i^s \leftarrow \infty$ for all $i \in N$; $Q_z^s \leftarrow \emptyset$ for $z = 1, ..., lmax$;
        $Q_0^s \leftarrow \{s\}$, $d_s^s \leftarrow 0, p_s^s \leftarrow s$, $T^s \leftarrow \emptyset$;
      synchronize processors
      while $flag = 0$ do
        let $z$ be the next index such that $Q_z^s \neq \emptyset$ ;
        for each $u \in Q_z^s$ do
          $Q_z^s \leftarrow Q_z^s \backslash \{u\}$;
          for each $(u, v) \in FS(u)$ such that $d_u^s + l_{uv} < d_v^s$ do
            $a \leftarrow d_v^s(\text{mod } lmax + 1)$;
            $d_v^s \leftarrow d_u^s + l_{uv}$;
            $b \leftarrow d_v^s(\text{mod } lmax + 1)$;
            $p_v^s \leftarrow u$;
            $Q_a^s \leftarrow Q_a^s \backslash \{v\}$;
            $Q_b^s \leftarrow Q_b^s \cup \{v\}$;
          end
          $T^s \leftarrow T^s \cup \{u\}$;
          if $u \in T^t$ then $flag \leftarrow 1$;
        end
      endwhile
      $\beta_s \leftarrow \min\{d_i^s + d_i^t : i \in T^s\}$;
    endif
    if $procid = 2$ then
      initialize:
        $p_i^t \leftarrow 0, d_i^t \leftarrow \infty$ for all $i \in N$; $Q_z^t \leftarrow \emptyset$ for $z = 1, ..., lmax$;
        $Q_0^t \leftarrow \{t\}$, $d_t^t \leftarrow 0, p_t^t \leftarrow t$, $T^t \leftarrow \emptyset$;
      synchronize processors
      while $flag = 0$ do
        let $z$ be the next index such that $Q_z^t \neq \emptyset$ ;
        for each $v \in Q_z^s$ do
          $Q_z^t \leftarrow Q_z^t \backslash \{v\}$;
          for each $(u, v) \in BS(v)$ such that $d_v^t + l_{uv} < d_u^t$ do
            $a \leftarrow d_u^t(\text{mod } lmax + 1)$;
            $d_u^t \leftarrow d_v^t + l_{uv}$;
            $b \leftarrow d_u^t(\text{mod } lmax + 1)$;
            $p_u^t \leftarrow v$;
            $Q_a^t \leftarrow Q_a^t \backslash \{u\}$;
            $Q_b^t \leftarrow Q_b^t \cup \{u\}$;
          end
          $T^t \leftarrow T^t \cup \{v\}$;
          if $v \in T^s$ then $flag \leftarrow 1$;
```

```
            end
         endwhile
         $\beta_t \leftarrow \min\{d_i^s + d_i^t : i \in T^t\}$;
      endif
      comment: end use of multiple processors
      join processors
      $\beta \leftarrow \min\{\beta_s, \beta_t\}$;
      $J \leftarrow \{i \in T^s \cup T^t : d_i^s + d_i^t = \beta\}$;
   end
```

1.9 The parallel two-tree S2 algorithm

As in the previous parallel algorithms, the parallel two-tree S2 algorithm assigns one processor to work on the tree rooted at s and another processor to work on the tree rooted at t. When a node first appears in both trees, a flag is set to initiate the mop-up node scanning phase. As explained in Section 1.6, we perform the mop-up scanning phase from the smaller tree only. To perform this work with two processors requires each one to share the same data, namely, distance labels. To prevent possible interference with each other, parallel processing constructs called *locks* are used so only one processor at a time may update a distance label. Following this, the processors are synchronized and the minimum doubly labelled nodes are found for each tree. The minimum of these two gives the minimum distance path from s to t and a shortest-path is implicit in the predecessor labels. The procedures SINSERT(x) and TINSERT(x) are as presented in Section 1.6. The algorithm may be stated as follows:

```
Procedure PS22($s,t$)
   begin
      $flag \leftarrow 0$;
      comment: begin use of two processors
      fork(2)
      if $procid = 1$ then
         initialize:
            $p_i^s \leftarrow 0, d_i^s \leftarrow \infty, fk(i) \leftarrow 1, fh(i) = |FS(i)|$ for all $i \in N$;
            $Q_z^s \leftarrow \emptyset$ for $z = 1, ..., lmax$; $Q_0^s \leftarrow \{s\}$, $d_s^s \leftarrow 0, p_s^s \leftarrow s$, $T^s \leftarrow \emptyset$;
            $p_i^t \leftarrow 0, d_i^t \leftarrow \infty, bk(i) \leftarrow 1, bh(i) = |BS(i)|$ for all $i \in N$;
            $Q_z^t \leftarrow \emptyset$ for $z = 1, ..., lmax$; $Q_0^t \leftarrow \{t\}$, $d_t^t \leftarrow 0, p_t^t \leftarrow t$, $T^t \leftarrow \emptyset$;
         synchronize processors
         while $flag = 0$ do
            let $z$ be the next index such that $Q_z^s \neq \emptyset$ ;
            for each $u \in Q_z^s$ do
               comment: determine next node in $FS(u)$
               $v \leftarrow w_{fk(u)}$;

               $Q_z^s \leftarrow Q_z^s \backslash \{u\}$;
               comment: determine new candidate arc
               SINSERT($u$);
               $T^s \leftarrow T^s \cup \{u\}$;
               if $v \notin Q^s$ then SINSERT($v$);
               if $u \in T^t$ then $flag \leftarrow 1$;
            end
         endwhile
      endif
   if procid = 2 then
      initialize:
         $p_i^t \leftarrow 0, d_i^t \leftarrow \infty, bk(i) \leftarrow 1, bh(i) = |BS(i)|$ for all $i \in N$;
         $Q_z^t \leftarrow \emptyset$ for $z = 1, ..., lmax$; $Q_0^t \leftarrow \{t\}$, $d_t^t \leftarrow 0, p_t^t \leftarrow t$, $T^t \leftarrow \emptyset$;
         synchronize processors
         while $flag = 0$ do
            let $z$ be the next index such that $Q_z^t \neq \emptyset$ ;
            for each $v \in Q_z^t$ do
            comment: determine next node in $BS(u)$
            $u \leftarrow w_{bk(v)}$;

            $Q_z^t \leftarrow Q_z^t \backslash \{v\}$;
            comment: determine new candidate arc
            TINSERT($v$);
            $T^t \leftarrow T^t \cup \{v\}$;
            if $u \notin Q^t$ then TINSERT($u$);
            if $v \in T^s$ then $flag \leftarrow 1$;
         end
      endwhile
   endif
   comment: mop-up phase from smaller tree
   synchronize processors
   if $|T^s| < |T^t|$ then
      comment: each processor works on next unscanned node $u \in T^s$
      for each $u \in T^s$ do
         for each $(u, y) \in FS(u)$ do
            if $y \in T^s \cup T^t$ then
               if $d_u^s + l_{uy} < d_y^s$ then
                  set locked
                     $d_y^s \leftarrow d_u^s + l_{uy}$;
                     $p_y^s \leftarrow u$;
                  set unlocked
               endif
            endif
         end
      end
   else
      for each $v \in T^t$ do
      comment: each processor works on next unscanned node $v \in T^t$
         for each $(w, v) \in BS(v)$ do
            if $w \in T^s \cup T^t$ then
               if $d_v^t + l_{wv} < d_w^t$ then
                  set locked
                     $d_w^t \leftarrow d_v^t + l_{wv}$;
                     $p_w^t \leftarrow v$;
                  set unlocked
               endif
            endif
         end
      end
   endif
   synchronize processors
   if $procid = 1, \beta_s \leftarrow \min\{d_i^s + d_i^t : i \in T^s\}$;
   if $procid = 2, \beta_t \leftarrow \min\{d_i^s + d_i^t : i \in T^t\}$;
   join processors
   $\beta \leftarrow \min\{\beta_s, \beta_t\}$;
   $J \leftarrow \{i \in T^s \cup T^t : d_i^s + d_i^t = \beta\}$;
end
```

2. COMPUTATIONAL EXPERIENCE

All nine algorithms have been coded in FORTRAN and run on a Sequent Symmetry S81 using either one or two Intel 80386 processors. Several factors affect the performance of shortest-path codes. First, the number of nodes is important for Dijkstra-type algorithms, whose majority of work is searching a node-length array for a minimum label node. Second, the average degree ($|A|/|N|$) is important because the Dijkstra-type and S1-type algorithms must scan entire forward (or backward) stars each iteration, while S2-type algorithms typically scan only a subset. Finally, the cost range of a network affects the length and sparseness of the Q array for S1-type and S2-type algorithms, which are searched each iteration. The cost range and degree also affect the number of nodes that tie with a minimum distance label, thereby reducing the number of searches.

Four node levels (1000, 2000, 3000, 4000), ten average degree levels (5, 10, 15, 20, 25, 50, 75, 100, 125, 150), and three cost ranges (1-100, 1-1000, 1-10000) were chosen as being varied enough to demonstrate which factors were influencing performance. The total number of random networks generated was 120. Each code solved twenty problems per network using the same randomly generated s and t nodes, yielding a total of 2400 problems. Each data point in Tables 1-3 is the sum of times (in seconds) to solve the twenty problems. Since the S2-type codes require sorted forward and backward stars, all codes were given sorted forward and backward stars to eliminate this as a relative factor among them. It is debatable whether or not the sorting time should be counted against the S2-type algorithms since it *requires* sorted arc-lists. Here we simply assume that the data is available in pre-sorted order and concede that if it were not available, the S2-type algorithms would not be appropriate.

With nine codes and 120 networks, many comparisons and observations can be made. We highlight the major points of interest. First, there is some overlap with previous studies and we wish to confirm previous results. As in Dial et al. (1979), we find that S1 is better than S2 on only the smallest degree problems. As the forward stars get larger, the savings of scanning only a subset are realized. On four high node number, low degree networks, Mohr and Pasche (1988) found that a D2-type algorithm required about 38% of the time needed by a D1 algorithm. Here we found that the averages for D2 were 65%, 49%, and 23% of that for D1 for the cost ranges 1-100, 1-1000, and 1-10000 respectively.

PS22, the parallel two-tree S2 code, is the overall winner. It had the fastest time on 108 out of the 120 networks, while PS12 was fastest on 12 networks. PS22 was the fastest code when the average degree increased above 10 on networks with 1–100 cost range and when the average degree increased above 5 on networks with 1–1000 cost range. It was always the fastest code when the cost range was 1–10000. PD2 improved on the networks with the lowest number of nodes, lowest cost, and highest degree — all factors that reduce the number of and time for searches for minimum label nodes by increasing ties. Were the degree increased to make these networks much more dense, PD2 might become more competitive.

Overall, S22 was the fastest sequential code, and was even faster than PS12 and PD2. It had the fastest time of the sequential codes on all 120 networks. In general, the time required using two S1-type trees is about 23%, 26%, and 36% of the time using only one S1 tree on the 1-100, 1-1000, and 1-10000 cost ranges respectively. Similarly, two S2-type trees require on average 23%, 27%, and 37% of the time required by the S2 code on 1-100, 1-1000, and 1-10000 cost range networks, respectively.

Dreyfus (1969) comments that savings may accrue for two-tree algorithms if the stopping criterion is reached well before $N/2$ nodes are permanently labelled in each tree. This is definitely the case for random networks. The one-tree algorithms scan approximately 50.4% of the nodes in the network until t is placed in the tree, while the two-tree algorithms scan only about 4.7% of the nodes until one is first placed in both trees. That is, two-tree algorithms scan about 9.3% of the nodes scanned by one-tree algorithms, resulting in the above mentioned savings in time.

It should be noted that we also solved the above problems using the efficient label-correcting code THRESH-X2 (see Glover, Klingman, Phillips, and Schneider (1985)). The total times in seconds for the 1-100, 1-1000, and 1-10000 cost range networks were 877.9, 963.4, and 976.0, respectively. Since this algorithm solves the one-to-all shortest-path problem, it was not included in the tables. We can see, however, that it is more efficient to use label-setting algorithms for the one-to-one problem since they stop before the entire tree is built.

When using two processors, D2 and S12 parallelize nicely as the PD2 and PS12 codes. PD2 averages a speed-up of 1.93 over D2. PS12 averages a speed-up of 1.93 over its sequential counterpart, S12. In fact, on some networks a speed-up over 2.0 is achieved. This is due to ties for the minimum label node. The sequential versions scan all nodes that tie in one tree before moving to the other tree. With two processors, a node that is first scanned from a group with ties could be the one that is placed next in the opposite tree and the remaining tied nodes need not be scanned as they are in the one processor version. Less work in parallel results in a speed-up over 2.0. If the sequential versions scanned a node from each tree alternatel·, the speed-up would be less than 2.0, but overall this was slightly slower than scanning all nodes that tied.

PS22 averages only a speed-up of 1.40 over S22. This lower speed-up is due to the additional cost of using the parallel processing *locks* during the relatively lengthy mop-up phase. That is, when one processor has *locked* a section of code, the other processor waits idly until the section becomes *unlocked* before it can execute the same section.

3. SHORTEST-PATH HEURISTICS

There are times when it may be of interest to quickly find "good" paths in a network. For example, when finding a starting solution to a network flow problem, paths may be found to send flow from sources to sinks that do not have to be minimum paths. "Good" paths at the start may mean the minimum cost flow will be found more quickly. We find there is a trade-off between the time to find a path and the length of the path relative to a minimum path.

We have seen in Section 2 that the S22 code is the overall fastest sequential code for finding a shortest-path between two nodes. Recall that this algorithm requires a mop-up phase to scan all remaining unscanned arcs in the forward or backward stars of the nodes in each shortest-path tree before a shortest-path can be found. This mop-up phase has been found to take from 3% to 70% of the total time for low degree to high degree networks, respectively. However, before this mop-up phase we have a node that is in both shortest-path trees and a path from s to t implicit in the predecessor labels. It may not be an optimal path, but it is likely to be good and is found much quicker on higher degree networks. The heuristic code H2 used the path implied by this first node in each shortest-path tree.

Following the mop-up phase in the optimal S22 code, there is a search over all nodes in each tree for the one with the minimum sum of its distance labels. This same search may be done at the end of heuristic H2, without doing the mop-up phase, to see if there is a better path than the one implied by the first node in both trees. Heuristic H3 is identical to H2, but performs this additional step.

Paths between s and t are known before a node appears in both shortest-path trees. Heuristic code H1 uses the path implied by the node that first has a finite distance label from both s and t.

Tables 4–6 show the times for twenty problems per network on the same networks used in testing the optimal algorithms. Also shown is the percentage this time was of the optimal S22 algorithm. As expected, substantial savings in time are realized on high degree networks, where the mop-up phase dominates the S22 times. On average, H1 requires about 65% of S22 time, while H2 and H3 require 73% and 75% respectively. However on average, H1 ranges from 81% of the S22 time on the lowest degree networks to 46% of S22 time on the highest degree networks. Similarly, H2 has an average range of 93% to 51% of S22 time and H3 has an average range of 95% to 52% of S22 time.

Tables 7–9 show how good these paths are compared to the actual shortest-paths. On average, H1 found the shortest-path 55% of the time. The average length of its path was 8% greater than the shortest-path and the worst path found averaged 44% greater than the shortest-path. H2 found 72% of the shortest-paths (reaffirming the necessity of the mop-up phase). Its average path length was 2.4% greater than the shortest-path and the worst path it found averaged 19% greater than the optimal path. H3 found 93% of the shortest-paths with an average path length 0.5% greater than the shortest-path. Its worst path averaged 5% greater than the shortest-path.

It should be noted that in a few instances, H1 found more of the optimal paths for a given network than H2. (See Table 10, nodes = 1000 and degree = 5.) This is similar to the case in which the first node placed in both shortest-path trees is not necessarily on a shortest-path. Here, the node that first has two finite labels (H1) is on a shortest-path, but is not the node that is first placed in both trees (H2).

4. CONCLUSION

The objective of this paper has been to present four new shortest-path algorithms, two sequential and two parallel, and to empirically compare them with five algorithms previously discussed in the literature. The new algorithms combine the highly effective data structures of the S1 and S2 algorithms with the idea of building trees from a source node and a sink node in order to find a shortest-path. We found that the new S22 algorithm was the fastest sequential algorithm on all networks. The new parallel algorithm, PS22, was the fastest algorithm on all but the lowest degree networks, where PS12 was the fastest. It appears that the parallel two-tree Dijkstra algorithm, PD2, might be competitive only on very low cost, dense networks.

The secondary topic of this paper is heuristic S22-type algorithms for obtaining near-minimum paths. Three new heuristic shortest-path algorithms were discussed and were shown to find very good (often optimal) paths from a source to a sink much faster than the shortest-path can be found. These heuristics eliminate the time-consuming mop-up phase required in the S22 algorithm and are quite effective on higher degree networks.

5. APPENDIX

Tables 1–3 present the computational results for solving twenty problems for each of the nine shortest-path algorithms discussed above. Tables 4–6 present the computational results for solving twenty problems for the three S2-type heuristics discussed above. Tables 7–9 show how often the heuristics found the optimal solution and how far off the solutions were when they did not.

Table 1. - - Time in seconds for 20 problems (Cost range: 1-100)

		code								
nodes	degree	D1	S1	S2	D2	S12	S22	PD2	PS12	PS22
1000	5	7.62	0.79	0.84	6.22	0.46	0.46	3.38	0.35	0.41
	10	5.03	1.30	0.94	5.06	0.61	0.54	2.79	0.42	0.43
	15	3.81	1.59	0.90	3.86	0.69	0.48	2.18	0.44	0.42
	20	3.65	2.06	1.25	3.64	0.87	0.55	2.07	0.53	0.46
	25	4.17	3.06	1.74	3.52	1.01	0.57	2.00	0.61	0.48
	50	4.03	4.57	2.05	2.50	1.50	0.65	1.49	0.82	0.53
	75	4.67	6.21	2.09	2.27	1.91	0.73	1.36	1.05	0.60
	100	6.65	9.00	4.01	2.50	2.63	0.89	1.36	1.30	0.64
	125	6.74	9.87	4.33	2.10	2.57	0.93	1.25	1.36	0.63
	150	9.46	13.72	5.75	2.53	3.32	1.11	1.40	1.70	0.72
2000	5	23.02	2.28	2.51	18.50	0.92	0.89	9.61	0.59	0.68
	10	12.88	3.23	2.83	12.98	1.13	0.93	6.80	0.69	0.70
	15	8.73	3.43	2.01	9.67	1.34	0.95	5.27	0.78	0.72
	20	9.63	5.50	3.32	8.88	1.55	0.93	4.82	0.83	0.73
	25	9.28	6.55	3.35	8.36	1.89	1.09	4.57	1.06	0.78
	50	10.05	11.14	5.31	5.97	2.87	1.21	3.33	1.52	0.92
	75	9.70	12.56	3.65	4.69	3.22	1.23	2.65	1.59	0.89
	100	13.37	18.91	8.79	4.35	4.00	1.41	2.42	2.18	0.98
	125	17.01	23.21	9.16	4.53	5.18	1.52	2.51	2.45	1.05
	150	18.43	27.33	8.74	4.81	5.91	1.79	2.62	2.89	1.15
3000	5	35.26	3.23	3.44	31.17	1.30	1.24	15.72	0.79	0.92
	10	20.38	5.00	3.98	19.70	1.55	1.25	10.38	0.90	0.94
	15	17.52	6.86	4.95	17.94	1.96	1.34	9.29	1.09	1.00
	20	14.28	8.14	5.26	13.39	2.22	1.37	7.13	1.21	0.99
	25	14.05	10.11	5.96	12.97	2.82	1.47	6.99	1.37	1 08
	50	11.71	12.90	4.56	8.30	3.37	1.53	4.60	1.71	1.11
	75	14.73	18.52	5.32	7.09	4.44	1.73	4.04	2.26	1.23
	100	20.02	27.96	9.09	7.14	5.94	1.96	3.99	3.04	1.28
	125	22.83	33.82	12.85	6.54	6.93	2.14	3.46	3.33	1.33
	150	23.88	34.91	12.52	5.91	6.49	2.14	3.33	3.27	1.32
4000	5	48.97	4.69	4.83	42.68	1.70	1.58	22.13	0.99	1.17
	10	27.81	6.26	5.42	26.38	2.00	1.59	13.73	1.11	1.18
	15	22.66	8.86	6.05	23.14	2.54	1.72	12.77	1.36	1.27
	20	20.28	11.21	7.03	19.05	2.68	1.71	10.11	1.45	1.28
	25	21.14	15.66	9.17	17.76	3.32	1.81	9.39	1.71	1.36
	50	18.55	20.51	8.52	11.65	4.49	1.99	6.46	2.18	1.40
	75	24.84	32.71	14.88	10.82	6.69	2.23	5.78	3.13	1.55
	100	27.68	38.53	13.72	9.12	6.84	2.37	5.03	3.24	1.56
	125	30.99	44.23	13.37	9.17	8.66	2.56	4.93	3.90	1.69
	150	30.05	47.07	10.90	8.34	9.63	2.70	4.44	4.39	1.76
total		655.56	557.49	235.39	425.20	129.15	55.29	227.58	65.59	39.34

Table 2. - - Time in seconds for 20 problems (Cost range: 1–1000)

		code								
nodes	degree	D1	S1	S2	D2	S12	S22	PD2	PS12	PS22
1000	5	30.08	1.16	1.15	8.70	0.78	0.67	4.62	0.51	0.54
	10	20.47	1.52	1.14	9.82	0.87	0.63	5.20	0.55	0.53
	15	20.95	2.28	1.45	9.37	0.98	0.63	4.97	0.62	0.54
	20	15.79	2.31	1.38	8.56	1.11	0.63	4.55	0.66	0.55
	25	14.87	3.11	1.64	7.60	1.15	0.63	4.07	0.69	0.55
	50	10.59	4.77	1.99	6.91	1.83	0.78	3.76	1.04	0.64
	75	10.74	7.26	2.73	7.04	2.50	0.94	3.71	1.26	0.69
	100	10.54	9.24	2.95	6.12	2.91	1.03	3.33	1.54	0.79
	125	11.78	11.68	4.75	6.22	3.58	1.23	3.36	1.80	0.89
	150	9.90	10.46	3.21	5.07	3.31	1.12	2.76	1.72	0.79
2000	5	93.53	2.19	2.33	27.70	1.21	1.06	14.14	0.75	0.80
	10	64.37	3.43	2.52	24.63	1.32	1.02	12.65	0.80	0.79
	15	50.13	4.35	3.12	23.35	1.53	1.06	12.18	0.89	0.80
	20	41.88	5.34	3.31	20.52	1.72	1.02	10.69	0.94	0.79
	25	42.08	7.38	4.78	21.40	2.07	1.11	10.95	1.08	0.84
	50	28.76	12.52	5.96	19.29	3.35	1.33	9.87	1.70	0.99
	75	22.46	14.42	6.12	14.36	3.85	1.40	7.62	1.90	1.03
	100	23.51	19.21	4.62	15.13	5.39	1.74	7.92	2.56	1.22
	125	21.25	20.12	5.61	11.78	5.02	1.68	6.25	2.55	1.14
	150	27.28	28.97	8.25	12.22	6.68	2.00	6.48	3.13	1.29
3000	5	157.41	3.44	3.52	52.47	1.67	1.44	26.95	0.99	1.09
	10	113.51	5.83	4.78	44.69	1.88	1.42	23.37	1.06	1.03
	15	84.83	6.93	4.14	43.59	2.14	1.40	21.86	1.18	1.07
	20	65.96	8.14	4.81	42.00	2.64	1.52	21.12	1.37	1.13
	25	63.72	10.61	5.97	39.18	2.83	1.50	19.74	1.46	1.13
	50	39.78	15.73	7.43	25.20	3.84	1.63	13.08	1.88	1.22
	75	41.44	26.88	11.87	27.83	5.98	2.06	14.06	2.85	1.38
	100	34.30	26.80	9.63	20.02	5.99	1.98	10.48	2.79	1.36
	125	34.88	32.14	13.63	18.56	6.55	2.05	9.72	3.15	1.39
	150	38.17	39.71	13.01	17.74	7.94	2.26	9.30	3.67	1.42
4000	5	269.68	4.98	5.42	78.45	2.03	1.81	39.54	1.17	1.32
	10	172.27	8.38	6.76	81.07	2.51	1.83	40.44	1.37	1.35
	15	105.57	7.47	4.21	52.22	2.40	1.67	26.15	1.33	1.28
	20	100.07	11.86	6.64	62.87	3.17	1.85	31.88	1.64	1.36
	25	61.74	12.56	6.36	56.04	3.36	1.89	28.03	1.76	1.41
	50	54.62	21.43	7.01	40.52	4.94	2.05	20.58	2.39	1.48
	75	51.40	31.54	8.64	36.33	6.52	2.36	18.53	3.26	1.65
	100	51.97	43.10	13.61	34.58	9.74	2.93	17.88	4.75	1.81
	125	39.31	33.45	7.67	26.70	8.98	2.84	13.79	4.43	1.84
	150	51.74	50.83	15.91	24.54	10.33	2.99	12.53	4.82	1.93
total		2223.33	573.53	230.03	1090.39	146.60	61.19	558.11	74.04	42.72

Table 3. - - Time in seconds for 20 problems (Cost range: 1–10000)

		code								
nodes	degree	D1	S1	S2	D2	S12	S22	PD2	PS12	PS22
1000	5	51.21	3.77	3.51	11.88	3.86	2.57	6.24	2.25	1.97
	10	42.60	3.36	2.77	9.03	3.01	1.81	4.80	1.81	1.49
	15	38.74	3.47	2.46	10.66	2.97	1.63	5.62	1.77	1.39
	20	40.55	3.96	2.58	10.70	2.92	1.52	5.62	1.75	1.32
	25	43.22	4.59	2.74	9.99	2.99	1.53	5.30	1.78	1.30
	50	30.62	5.83	2.53	9.21	3.34	1.42	4.88	1.95	1.24
	75	34.97	9.09	4.00	11.77	4.31	1.64	6.16	2.34	1.37
	100	28.98	10.11	3.87	9.32	4.28	1.63	4.93	2.37	1.36
	125	24.48	10.66	3.02	9.21	4.72	1.70	4.87	2.57	1.40
	150	25.63	12.82	3.79	8.10	4.79	1.66	4.33	2.57	1.39
2000	5	170.47	5.02	4.63	30.39	4.32	2.99	15.70	2.47	2.26
	10	152.29	5.32	4.31	25.82	3.57	2.27	13.52	2.11	1.77
	15	150.98	6.23	4.22	31.87	3.70	2.11	16.38	2.13	1.68
	20	129.11	6.69	4.15	33.01	3.88	2.03	16.81	2.21	1.67
	25	109.28	6.65	4.19	26.79	3.73	1.90	13.92	2.15	1.57
	50	101.63	12.15	5.21	28.23	4.87	2.01	14.45	2.64	1.60
	75	90.39	17.34	6.67	28.80	5.86	2.13	14.80	3.15	1.72
	100	66.55	17.62	5.15	27.10	6.51	2.27	14.11	3.43	1.69
	125	62.73	21.42	5.41	28.49	7.54	2.50	14.11	3.86	1.89
	150	65.88	28.84	8.73	27.35	8.46	2.69	14.11	4.32	2.01
3000	5	380.23	6.43	6.07	53.09	4.80	3.44	27.06	2.75	2.54
	10	330.84	7.48	5.82	49.91	4.15	2.67	25.16	2.37	2.05
	15	313.76	9.13	6.07	51.80	4.22	2.53	26.47	2.40	1.95
	20	309.35	11.13	7.24	50.05	4.33	2.37		2.47	1.89
	25	213.96	10.24	5.24	47.35	4.42	2.26	24.08	2.50	1.86
	50	165.98	16.37	6.01	52.71	6.07	2.43	26.91	3.25	1.93
	75	155.82	26.33	13.15	59.12	8.16	2.82	30.14	4.29	2.12
	100	127.05	29.77	12.26	51.31	8.86	2.92	26.22	4.57	2.16
	125	81.45	24.72	7.66	35.55	7.83	2.54	17.83	4.11	1.90
	150	106.22	41.81	10.07	41.62	9.84	3.00	20.97	5.04	2.15
4000	5	741.87	8.62	8.65	108.49	5.51	4.02	53.39	3.12	2.94
	10	386.83	7.57	5.51	61.50	4.33	2.92	30.72	2.50	2.24
	15	527.01	12.33	8.74	89.12	4.90	2.90	44.47	2.78	2.27
	20	416.12	13.36	7.33	82.19	5.03	2.75	41.75	2.78	2.15
	25	381.66	15.51	8.32	67.48	4.97	2.64	33.89	2.78	2.07
	50	233.50	21.68	6.57	81.93	7.14	2.86	40.43	3.81	2.25
	75	199.77	29.65	8.75	68.66	8.04	2.91	34.36	4.26	2.24
	100	187.13	43.26	19.67	69.30	9.82	3.19	34.60	5.01	2.38
	125	135.70	39.82	11.15	56.98	9.93	3.15	28.98	5.16	2.30
	150	131.56	47.19	14.34	61.04	12.08	3.68	30.88	6.03	2.60
total		6986.12	617.34	262.56	1626.92	224.06	98.01	824.07	121.61	76.08

Table 4. - - Heuristic times in seconds for 20 problems (Cost range: 1-100)

			degree											overall
code	nodes		5	10	15	20	25	50	75	100	125	150	avg	avg
H1	1000	time	0.33	0.35	0.34	0.34	0.35	0.34	0.34	0.37	0.34	0.34	0.34	
		% of S22	71.2	63.8	70.1	62.2	60.9	52.3	47.5	41.0	36.7	31.0	53.7	
	2000	time	0.69	0.68	0.67	0.66	0.68	0.68	0.66	0.67	0.68	0.67	0.67	
		% of S22	77.3	72.9	70.5	71.6	62.6	56.3	53.4	47.7	44.5	37.5	59.4	0.83
	3000	time	1.03	1.00	0.99	0.98	1.00	0.98	0.98	1.00	1.00	0.98	0.99	61.7
		% of S22	83.0	80.3	74.3	70.9	68.4	64.0	56.5	51.1	47.0	45.9	66.7	
	4000	time	1.33	1.28	1.30	1.31	1.30	1.29	1.30	1.31	1.31	1.34	1.31	
		% of S22	83.9	80.4	75.5	76.7	71.8	65.2	58.4	55.3	51.3	49.6	66.8	
H2	1000	time	0.40	0.42	0.38	0.41	0.41	0.39	0.39	0.41	0.40	0.40	0.40	
		% of S22	86.3	76.4	79.0	74.8	71.4	60.2	53.4	46.2	42.7	36.4	62.7	
	2000	time	0.81	0.77	0.79	0.76	0.80	0.79	0.74	0.75	0.75	0.76	0.77	
		% of S22	90.5	82.9	83.8	81.8	73.1	65.2	60.0	52.9	49.0	42.3	68.2	0.93
	3000	time	1.15	1.12	1.14	1.11	1.14	1.10	1.09	1.11	1.10	1.08	1.11	69.3
		% of S22	92.4	89.4	85.6	80.9	77.5	71.8	63.3	56.6	51.4	50.3	71.9	
	4000	time	1.48	1.44	1.47	1.46	1.48	1.43	1.44	1.43	1.42	1.46	1.45	
		% of S22	93.7	90.7	85.7	85.1	81.5	72.2	64.5	60.4	55.5	54.1	74.3	
H3	1000	time	0.42	0.44	0.40	0.43	0.43	0.41	0.41	0.44	0.41	0.42	0.42	
		% of S22	90.6	80.5	83.2	79.4	74.7	62.7	56.7	48.8	44.7	37.9	65.9	
	2000	time	0.83	0.81	0.83	0.79	0.84	0.81	0.76	0.77	0.77	0.79	0.80	
		% of S22	93.2	86.5	87.3	85.5	76.6	67.2	61.9	54.7	50.8	43.8	70.8	0.97
	3000	time	1.18	1.19	1.18	1.15	1.18	1.17	1.12	1.18	1.12	1.11	1.16	71.9
		% of S22	94.9	95.1	88.0	83.4	80.4	76.3	65.0	60.1	52.5	51.7	74.7	
	4000	time	1.52	1.47	1.51	1.49	1.51	1.47	1.47	1.47	1.46	1.49	1.49	
		% of S22	96.1	92.7	87.7	87.1	83.3	73.9	66.0	61.9	56.9	55.3	76.1	

Table 5.- - Heuristic times in seconds for 20 problems (Cost range: 1-1000)

			degree											overall
code	nodes		5	10	15	20	25	50	75	100	125	150	avg	avg
H1	1000	time	0.49	0.44	0.43	0.40	0.40	0.42	0.41	0.40	0.41	0.40	0.42	0.92 61.4
		% of S22	73.7	70.0	69.0	64.0	62.6	53.5	43.1	38.7	33.2	35.4	54.3	
	2000	time	0.86	0.80	0.75	0.76	0.76	0.77	0.73	0.75	0.72	0.75	0.77	
		% of S22	80.7	78.6	70.9	74.0	68.3	57.9	52.3	43.0	42.7	37.4	60.6	
	3000	time	1.17	1.13	1.08	1.06	1.09	1.06	1.07	1.03	1.06	1.03	1.08	
		% of S22	81.1	79.5	77.2	69.7	72.7	64.9	51.9	51.9	51.5	45.6	64.6	
	4000	time	1.50	1.42	1.39	1.41	1.37	1.38	1.37	1.40	1.37	1.42	1.40	
		% of S22	82.8	77.6	83.5	76.4	72.5	67.3	58.1	47.8	48.0	47.6	66.2	
H2	1000	time	0.59	0.53	0.51	0.49	0.46	0.47	0.48	0.47	0.49	0.45	0.49	1.04 70.0
		% of S22	87.7	84.8	81.1	77.2	72.8	60.9	50.9	45.5	40.0	40.5	64.1	
	2000	time	0.98	0.90	0.87	0.85	0.86	0.88	0.84	0.88	0.81	0.85	0.87	
		% of S22	92.7	88.9	82.2	83.3	76.8	66.1	59.7	50.4	48.3	42.6	69.1	
	3000	time	1.34	1.26	1.22	1.23	1.22	1.17	1.23	1.15	1.14	1.14	1.21	
		% of S22	93.1	88.4	87.2	80.9	81.6	71.7	59.5	58.1	55.8	50.6	72.7	
	4000	time	1.68	1.65	1.50	1.58	1.55	1.52	1.53	1.61	1.52	1.57	1.57	
		% of S22	92.8	90.3	89.8	85.4	81.8	74.2	64.7	54.9	53.3	52.5	74.0	
H3	1000	time	0.61	0.56	0.52	0.51	0.48	0.50	0.51	0.49	0.52	0.48	0.52	1.07 72.3
		% of S22	91.3	88.7	83.1	80.0	76.1	64.2	53.7	48.0	42.2	42.6	67.0	
	2000	time	1.01	0.97	0.90	0.88	0.88	0.90	0.90	0.91	0.84	0.88	0.91	
		% of S22	95.5	95.9	85.0	85.8	79.0	67.9	64.3	52.4	50.1	44.3	72.0	
	3000	time	1.37	1.29	1.25	1.27	1.26	1.20	1.25	1.18	1.17	1.17	1.24	
		% of S22	95.2	90.4	89.0	83.2	84.1	73.5	60.8	59.3	57.0	51.9	74.4	
	4000	time	1.71	1.69	1.53	1.62	1.58	1.55	1.56	1.64	1.55	1.60	1.60	
		% of S22	94.5	92.3	91.8	87.4	83 6	75.7	66.1	56.1	54.4	53.6	75.6	

Table 6.- - Heuristic times in seconds for 20 problems (Cost range: 1-10000)

			degree											overall
code	nodes		5	10	15	20	25	50	75	100	125	150	avg	avg
H1	1000	time	2.05	1.54	1.32	1.23	1.21	1.05	1.04	1.03	1.00	1.00	1.25	
		% of S22	79.8	85.2	81.2	80.6	79.2	74.1	63.7	63.3	59.0	60.4	72.6	
	2000	time	2.59	1.86	1.71	1.61	1.53	1.42	1.36	1.34	1.35	1.33	1.61	
		% of S22	86.9	82.0	80.8	79.1	80.5	70.6	64.0	59.0	53.9	49.6	70.6	1.75
	3000	time	2.97	2.26	2.07	1.98	1.81	1.74	1.68	1.69	1.63	1.65	1.95	72.0
		% of S22	86.5	84.5	81.9	83.3	80.1	71.5	59.6	57.8	64.0	54.9	72.4	
	4000	time	3.32	2.53	2.42	2.22	2.19	2.02	1.37	1.98	1.95	2.02	2.20	
		% of S22	82.7	86.6	83.4	80.8	82.9	70.7	58.1	62.1	61.8	54.8	72.4	
H2	1000	time	2.46	1.72	1.51	1.37	1.33	1.15	1.16	1.10	1.07	1.05	1.39	
		% of S22	95.8	95.2	92.6	90.4	86.9	80.6	70.7	67.4	63.3	63.3	80.6	
	2000	time	2.90	2.13	1.94	1.83	1.69	1.55	1.51	1.47	1.48	1.46	1.80	
		% of S22	97.4	93.7	92.0	89.8	88.6	77.3	71.0	65.0	59.0	54.4	78.8	1.94
	3000	time	3.32	2.54	2.30	2.16	2.00	1.91	1.92	1.86	1.72	1.77	2.15	79.8
		% of S22	96.5	94.8	91.2	91.2	88.6	78.7	68.1	63.6	67.8	58.9	79.9	
	4000	time	3.87	2.76	2.68	2.49	2.36	2.24	1.53	2.14	2.08	2.19	2.43	
		% of S22	96.3	94.4	92.4	90.4	89.7	78.3	64.7	67.0	66.1	59.6	79.9	
H3	1000	time	2.48	1.75	1.53	1.40	1.35	1.16	1.19	1.13	1.09	1.08	1.42	
		% of S22	96.7	96.6	94.2	92.0	88.4	82.0	72.6	69.0	64.4	64.7	82.1	
	2000	time	2.94	2.16	1.97	1.86	1.71	1.58	1.59	1.50	1.51	1.50	1.83	
		% of S22	98.6	94.9	93.5	91.4	89.8	78.7	74.6	66.1	60.2	55.6	80.3	1.98
	3000	time	3.35	2.60	2.34	2.20	2.07	1.99	1.96	1.89	1.79	1.84	2.20	81.4
		% of S22	97.5	97.1	92.5	92.6	91.6	81.8	69.6	64.7	70.2	61.2	81.9	
	4000	time	3.91	2.78	2.74	2.52	2.39	2.28	1.56	2.18	2.11	2.22	2.47	
		% of S22	97.3	95.3	94.6	91.7	90.7	79.6	66.1	68.3	67.0	60.5	81.1	

Table 7.- - Solution data for 20 problems (Cost range: 1-100)

			degree											overall
code	nodes		5	10	15	20	25	50	75	100	125	150	avg	avg
H1	1000	% opt	85.0	55.0	55.0	45.0	70.0	65.0	60.0	65.0	70.0	95.0	66.5	
		%>opt	2.6	11.7	10.9	11.8	6.5	9.4	7.8	4.7	6.0	1.2	7.3	
		worst %	38.6	57.9	48.4	45.9	50.0	42.3	26.7	30.0	37.5	25.0	40.2	
	2000	% opt	50.0	55.0	50.0	50.0	40.0	65.0	60.0	80.0	70.0	40.0	56.0	
		%>opt	9.1	7.0	10.7	10.7	8.4	2.9	7.6	4.2	5.6	12.8	7.9	58.3
		worst %	44.8	30.5	44.6	71.0	50.0	12.5	28.6	26.7	30.0	45.5	38.4	7.6
	3000	% opt	55.0	60.0	40.0	60.0	40.0	65.0	55.0	45.0	70.0	65.0	55.5	39.5
		%>opt	4.6	9.1	9.4	10.3	12.6	9.3	7.2	7.1	4.8	5.6	8.0	
		worst %	29.5	94.5	35.1	53.5	46.8	41.7	25.0	44.4	27.3	22.2	42.0	
	4000	% opt	40.0	70.0	35.0	65.0	50.0	60.0	65.0	65.0	50.0	50.0	55.0	
		%>opt	12.8	3.7	9.8	3.4	6.5	4.8	4.8	5.2	11.0	11.0	7.3	
		worst %	36.2	22.9	46.6	21.7	43.9	20.0	25.0	50.0	50.0	57.1	37.3	
H2	1000	% opt	75.0	85.0	85.0	70.0	80.0	65.0	80.0	95.0	80.0	85.0	80.0	
		%>opt	3.7	1.0	2.2	3.1	1.3	5.0	3.1	0.5	4.8	3.0	2.8	
		worst %	38.6	8.2	26.2	14.3	8.1	23.5	26.7	9.1	33.3	28.6	21.7	
	2000	% opt	65.0	80.0	70.0	80.0	60.0	75.0	85.0	90.0	85.0	75.0	76.5	
		%>opt	2.4	1.7	2.7	1.2	3.1	2.7	1.4	0.8	2.0	4.7	2.3	76.1
		worst %	16.8	22.1	13.8	11.3	16.7	17.6	13.3	12.5	25.0	42.9	19.2	2.6
	3000	% opt	75.0	60.0	65.0	70.0	70.0	60.0	90.0	75.0	85.0	80.0	73.0	19.9
		%>opt	1.8	3.6	2.7	2.9	2.3	5.0	0.7	2.9	1.4	2.8	2.6	
		worst %	16.7	35.0	15.7	19.1	15.2	37.5	6.3	15.4	9.1	25.0	19.5	
	4000	% opt	65.0	55.0	65.0	85.0	70.0	90.0	75.0	75.0	85.0	85.0	75.0	
		%>opt	4.1	4.5	2.6	1.0	2.3	1.0	2.4	2.8	2.4	1.7	2.5	
		worst %	35.5	13.7	20.0	7.0	15.2	12.0	40.0	20.0	16.7	11.1	19.1	
H3	1000	% opt	100.0	90.0	100.0	95.0	100.0	100.0	85.0	100.0	95.0	100.0	96.5	
		%>opt	0.0	0.7	0.0	0.1	0.0	0.0	1.2	0.0	0.6	0.0	0.3	
		worst %	0.0	7.1	0.0	1.8	0.0	0.0	7.7	0.0	10.0	0.0	2.7	
	2000	% cpt	100.0	80.0	95.0	100.0	75.0	100.0	90.0	95.0	95.0	90.0	92.0	
		%>opt	0.0	1.7	0.5	0.0	1.8	0.0	1.1	0.4	1.0	1.2	0.8	93.6
		worst %	0.0	22.1	7.2	0.0	12.1	0.0	13.3	5.6	25.0	9.1	9.4	0.6
	3000	% opt	95.0	95.0	95.0	95.0	90.0	85.0	100.0	90.0	90.0	95.0	93.0	7.8
		%>opt	0.1	0.4	0.5	0.7	0.3	2.4	0.0	1.3	1.0	0.6	0.7	
		worst %	2.1	6.5	6.2	8.8	2.0	37.5	0.0	14.3	9.1	9.1	9.6	
	4000	% opt	90.0	95.0	80.0	95.0	100.0	95.0	100.0	95.0	90.0	90.0	93.0	
		%>opt	0.9	0.4	0.9	0.2	0.0	0.7	0.0	0.8	1.4	1.2	0.7	
		worst %	11.2	9.6	9.6	4.3	0.0	12.0	0.0	20.0	18.2	11.1	9.6	

Table 8.- - Solution data for 20 problems (Cost range: 1-1000)

			degree											overall
code	nodes		5	10	15	20	25	50	75	100	125	150	avg	avg
H1	1000	% opt	55.0	60.0	35.0	60.0	55.0	50.0	55.0	45.0	40.0	50.0	50.5	
		%>opt	6.6	8.1	12.5	10.9	4.8	11.2	12.8	6.3	12.9	10.4	9.7	
		worst %	25.7	44.7	64.4	165.8	20.3	40.7	129.0	35.2	73.7	70.7	67.0	
	2000	% opt	55.0	70.0	35.0	70.0	50.0	50.0	55.0	45.0	65.0	40.0	53.5	
		%>opt	4.9	5.5	10.0	4.2	5.3	11.9	5.6	8.4	11.3	13.1	8.0	52.6
		worst %	19.5	74.1	34.3	31.8	37.4	57.1	23.4	38.2	73.7	82.8	47.2	8.2
	3000	% opt	45.0	55.0	70.0	70.0	50.0	75.0	35.0	75.0	65.0	40.0	58.0	46.9
		%>opt	10.8	3.2	2.7	5.3	8.4	4.3	8.6	4.0	5.4	8.1	6.1	
		worst %	48.4	14.3	20.7	35.4	31.4	23.7	29.1	21.6	46.5	34.3	30.5	
	4000	% opt	25.0	25.0	65.0	45.0	60.0	50.0	60.0	35.0	50.0	70.0	48.5	
		%>opt	10.9	11.3	7.4	10.4	7.8	7.2	6.5	13.1	7.7	5.8	8.8	
		worst %	39.1	36.4	68.6	24.3	53.0	26.9	40.4	38.4	61.3	41.0	42.9	
H2	1000	% opt	65.0	60.0	65.0	65.0	75.0	70.0	65.0	80.0	85.0	70.0	70.0	
		%>opt	2.8	5.7	4.5	4.9	2.3	1.8	4.1	1.4	0.8	4.9	3.3	
		worst %	38.0	32.9	49.4	49.8	31.5	12.3	12.8	7.7	8.2	31.3	27.4	
	2000	% opt	70.0	65.0	55.0	70.0	80.0	65.0	70.0	60.0	90.0	75.0	70.0	
		%>opt	2.0	1.4	2.3	2.4	0.8	3.4	2.3	3.9	0.4	1.1	2.0	69.9
		worst %	14.6	17.4	13.7	15.8	12.5	22.4	17.2	19.1	3.8	7.2	14.4	2.4
	3000	% opt	65.0	65.0	75.0	65.0	75.0	75.0	70.0	70.0	80.0	85.0	72.5	20.7
		%>opt	2.9	1.8	2.6	3.2	1.0	1.7	1.4	2.3	1.5	0.9	1.9	
		worst %	27.3	15.4	19.9	41.7	20.3	12.2	13.0	30.3	11.5	9.3	20.1	
	4000	% opt	65.0	45.0	70.0	60.0	65.0	70.0	80.0	60.0	70.0	85.0	67.0	
		%>opt	1.8	3.7	5.4	2.5	2.3	1.8	0.4	3.7	2.9	0.2	2.5	
		worst %	14.9	25.9	59.2	18.9	29.0	4.8	5.2	26.4	21.3	1.3	20.7	
H3	1000	% opt	100.0	100.0	95.0	90.0	95.0	85.0	100.0	95.0	95.0	85.0	94.0	
		%>opt	0.0	0.0	0.1	0.5	0.1	0.9	0.0	0.2	0.2	2.8	0.5	
		worst %	0.0	0.0	3.0	8.4	1.2	13.9	0.0	3.9	4.6	31.3	6.6	
	2000	% opt	80.0	95.0	95.0	90.0	95.0	85.0	95.0	95.0	100.0	90.0	92.0	
		%>opt	0.8	0.0	0.2	0.4	0.1	2.5	0.1	0.9	0.0	0.6	0.6	92.1
		worst %	8.4	0.6	5.2	4.5	1.3	20.7	1.9	19.1	0.0	7.2	6.9	0.6
	3000	% opt	90.0	95.0	90.0	90.0	95.0	100.0	90.0	95.0	95.0	100.0	94.0	7.6
		%>opt	1.6	0.0	1.5	0.2	0.8	0.0	0.4	0.1	0.2	0.0	0.5	
		worst %	27.3	1.2	19.9	3.1	20.3	0.0	5.0	1.2	5.2	0.0	8.3	
	4000	% opt	80.0	80.0	90.0	90.0	95.0	70.0	95.0	95.0	95.0	95.0	88.5	
		%>opt	0.8	0.7	0.6	0.2	0.0	1.4	0.0	1.0	0.8	0.1	0.6	
		worst %	4.3	8.8	10.4	2.0	0.3	4.8	0.9	21.8	28.9	1.3	8.4	

Table 9.- - Solution data for 20 problems (Cost range:1-10000)

			degree											overall
code	nodes		5	10	15	20	25	50	75	100	125	150	avg	avg
H1	1000	% opt	40.0	65.0	45.0	55.0	50.0	50.0	45.0	75.0	45.0	60.0	50.5	
		%>opt	8.4	12.7	9.8	7.3	6.9	14.3	11.8	3.2	7.7	6.8	9.7	
		worst %	32.0	90.7	40.6	35.9	54.9	89.5	48.1	26.2	21.9	59.4	67.0	
	2000	% opt	45.0	55.0	45.0	60.0	40.0	35.0	70.0	65.0	55.0	45.0	53.5	
		%>opt	5.4	6.2	6.3	13.3	13.8	11.6	8.1	6.2	11.2	9.3	8.0	52.6
		worst %	20.5	30.8	29.6	57.0	35.1	52.3	55.8	46.0	40.0	18.5	47.2	8.2
	3000	% opt	50.0	70.0	50.0	65.0	45.0	65.0	50.0	40.0	65.0	50.0	58.0	46.9
		%>opt	9.1	5.4	7.5	9.6	12.8	5.7	13.4	12.6	9.0	6.9	6.1	
		worst %	58.8	22.2	52.6	52.0	65.3	42.6	80.0	56.5	99.6	39.0	30.5	
	4000	% opt	40.0	55.0	65.0	50.0	30.0	35.0	60.0	60.0	60.0	60.0	48.5	
		%>opt	8.5	7.1	2.9	8.3	13.8	8.5	6.5	4.9	8.9	5.3	8.8	
		worst %	45.4	35.3	10.5	34.5	32.4	35.4	40.4	20.0	32.3	28.7	42.9	
H2	1000	% opt	60.0	85.0	55.0	80.0	45.0	80.0	65.0	75.0	75.0	80.0	70.0	
		%>opt	3.3	0.7	2.5	1.1	3.9	1.9	2.3	0.7	2.3	1.3	3.3	
		worst %	12.3	7.4	13.3	11.1	16.2	20.8	27.5	3.5	11.6	11.1	27.4	
	2000	% opt	80.0	75.0	50.0	70.0	80.0	75.0	75.0	65.0	65.0	70.0	70.0	
		%>opt	1.2	1.2	3.0	4.9	3.3	1.5	1.6	3.2	5.6	2.2	2.0	69.9
		worst %	12.5	10.0	17.4	30.6	31.8	10.9	15.9	21.6	65.9	23.5	14.4	2.4
	3000	% opt	75.0	75.0	65.0	90.0	80.0	80.0	65.0	70.0	100.0	65.0	72.5	20.7
		%>opt	2.1	2.2	3.5	0.3	1.7	1.4	4.2	2.5	0.0	1.5	1.9	
		worst %	11.7	20.4	21.1	4.4	11.4	10.3	33.1	17.6	0.0	14.3	20.1	
	4000	% opt	60.0	75.0	70.0	65.0	55.0	65.0	80.0	70.0	70.0	70.0	67.0	
		%>opt	1.4	2.5	2.3	2.7	2.8	2.7	0.4	2.4	2.4	2.8	2.5	
		worst %	5.9	27.1	16.3	15.9	9.3	14.7	5.2	19.2	17.5	34.7	20.7	
H3	1000	% opt	85.0	95.0	85.0	95.0	85.0	85.0	95.0	100.0	95.0	90.0	94.0	
		%>opt	0.8	0.3	0.4	0.2	0.4	0.8	0.1	0.0	0.0	0.2	0.5	
		worst %	6.1	5.2	4.5	2.7	2.4	6.3	1.5	0.0	0.3	2.2	6.6	
	2000	% opt	100.0	90.0	75.0	80.0	90.0	95.0	100.0	95.0	90.0	95.0	92.0	
		%>opt	0.0	0.1	1.2	3.1	0.7	0.4	0.0	0.2	0.4	0.2	0.6	92.1
		worst %	0.0	1.6	6.4	18.5	6.5	7.9	0.0	3.0	7.4	2.6	6.9	0.6
	3000	% opt	85.0	100.0	100.0	95.0	95.0	100.0	100.0	95.0	100.0	90.0	94.0	7.6
		%>opt	0.5	0.0	0.0	0.0	0.4	0.0	0.0	0.6	0.0	0.2	0.5	
		worst %	6.8	0.0	0.0	0.6	7.0	0.0	0.0	13.3	0.0	1.7	8.3	
	4000	% opt	95.0	90.0	95.0	85.0	100.0	95.0	95.0	90.0	90.0	95.0	88.5	
		%>opt	0.1	0.0	0.2	1.3	0.0	0.4	0.0	0.9	1.2	0.4	0.6	
		worst %	2.4	0.2	3.4	15.9	0.0	11.4	0.9	12.8	17.5	9.0	8.4	

REFERENCES

C. Berge and A. Ghouila, *Programming, Games, and Transportation Networks*, John Wiley and Sons, New York, NY (1962).

D. Bertsekas and R. Gallager, *Data Networks*, Prentice-Hall, Englewood Cliffs, NJ (1987).

G. Dantzig, "On the Shortest Route Through a Network," *Management Science*, 6 (1960) 187–190.

G. Dantzig, *Linear Programming and Extensions*, Princeton University Press, Princeton, NJ, 1963.

N. Deo and C. Pang, "Shortest-Path Algorithms: Taxonomy and Annotation," *Networks*, **14** (1984) 275

M. Desrochers, "A Note on the Partitioning Shortest Path Algorithm," *Operations Research Lette* (1987) 183–187.

R. Dial, "Algorithm 360: Shortest Path Forest With Topological Ordering," *Communications of the A* **12** (1969) 632–633.

R. Dial, F. Glover, D. Karney, and D. Klingman, "A Computational Analysis of Alternative Algorithms Labeling Techniques for Finding Shortest Path Trees," CCS Report 291, Center for Cybernetic Studies, University of Texas, Austin, TX 78712 (1977).

R. Dial, F. Glover, D. Karney, and D. Klingman, "A Computational Analysis of Alternative Algorithms Labeling Techniques for Finding Shortest Path Trees," *Networks*, 9 (1979) 215–250.

E. Dijkstra, "A Note on Two Problems in Connexion With Graphs," *Numerische Mathematik*, 1 (1 269–271.

J. Divoky, "Improvements for the Thresh X2 Shortest Path Algorithm," *Operations Research Lette* (1987) 227–232.

S. Dreyfus, "An Appraisal of Some Shortest-Path Algorithms," *Operations Research*, **17** (1969) 395–41

S. Even, *Graph Algorithms*, Computer Science Press, Potomic, MD (1979).

G. Gallo, and S. Pallottino, "Shortest Path Methods: A Unifying Approach," *Mathematical Programi Study*, **26** (1986) 38–64.

G. Gallo, and S. Pallottino, "Shortest Path Algorithms," *Annals of Operations Research*, **13** (1988) 3–

F. Glover, R. Glover, and D. Klingman, "Computational Study of an Improved Shortest Path Algorit *Networks*, **14** (1984) 25–36.

F. Glover, D. Klingman, N. Phillips, and R. Schneider, "New Polynomial Shortest Path Algorithms and T Computational Attributes," *Management Science*, **31**(1985) 1106–1128.

P. Hart, N. Nilsson, and B. Raphael, "A Formal Basis for the Heuristic Determination of Minimum Paths," *IEEE Transactions of Systems Science and Cybernetics*, SSC-4, (1968) 100–107.

T. Hu, *Combinatorial Algorithms*, Addison-Wesley, Reading, MA (1982).

P. Jensen and J. Barnes, *Network Flow Programming*, John Wiley and Sons, Inc., New York, NY (198

D. Klingman, J. Mote, and D. Whitman, "Improving Flow Management and Control Via Improving Sho Path Analysis," CCS Report 322, Center for Cybernetic Studies, The University of Texas, Austin, TX 78 (1978).

D. Klingman, A. Napier, and J. Stutz, "NETGEN: A Program for Generating Large Scale Capacit Assignment, Transportation, and Minimal Cost Flow Network Problems," *Management Science*, **20** (1 814–821.

E. Lawler, *Combinatorial Optimization: Networks and Matroids*, Holt, Rinehart, and Winston, New Y NY (1987).

T. Mohr and C. Pasche, "A Parallel Shortest Path Algorithm," *Computing*, **40** (1988) 281–292.

T. Nicholson, "Finding the Shortest Route Between Two Points in a Network," *The Computer Journ* (1966) 275–280.

C. Papadimitriou, and K. Steiglits, *Combinatorial Optimization: Algorithms and Complexity*, Prentice-I Englewood Cliffs, NJ (1987).

I. Pohl, “Bi-directional and Heuristic Search in Path Problems,” *SLAC Report No. 104*, Stanford, CA (

I. Pohl, “Bi-directional Search,” *Machine Intelligence*, 6, B. Meltzer and D. Michie, eds., Edinburgh Uni
Press, Edinburgh (1971) 127-140.

M. Quinn, *Designing Efficient Algorithms for Parallel Computers*, McGraw-Hill, New York, NY (198

R. Rockafellar, *Network Flows and Monotropic Optimization*, John Wiley and Sons, Inc., New Yor
(1984).

R. Tarjan, *Data Structures and Network Algorithms*, Society for Industrial and Applied Mather
Philadelphia, PA (1983).

APPENDIX C

Technical Report 93-CSE-7

A Direct Simplex Algorithm for Network Flow Problems with Piecewise Linear Costs

by

Rajluxmi V. Murthy
Richard V. Helgason

Department of Computer Science and Engineering
Southern Methodist University
Dallas, Texas 75275-0122

January 1993

This work was supported in part by the Air Force Office of Scientific Research und Grant Number AFOSR F49620-92-J-0032, and the Office of Naval Research under Grant Number N00014-92-1619.

A Direct Simplex Algorithm for Network Flow Problems with Piecewise Linear Co

Rajluxmi V. Murthy and **Richard V. Helgason**

Southern Methodist University

January, 1993

ABSTARCT. Minimum cost network flow problems with a piecewise linear convex cost function are ι model various optimization problems. They are also used extensively to approximate nonlinear cost fu which may otherwise be difficult to handle. Solving the piecewise linear problems using a reform approach is possible but may be inefficient. In this paper we discuss a specialized *direct* approach ; implementation for solving such problems. The *direct* approach handles the piecewise linear structure cost function by allowing each arc to have varying costs on different segments. Computational result been reported, from which we conclude that using such an approach has a distinct advantage over u reformulation approach.

1. Introduction

Minimum cost network flow problems with a convex piecewise linear cost function ($PLNP$) occur nat when costs change abruptly. This may be due to the imposition of penalties when certain limits or ‹ are exceeded by the flows in the arcs (see Rockafellar[1984]). For example, penalties can be impos understocked or oversupplied goods or an increase in transportation cost due to the load exceeding a sp limit. Random demand in a transshipment network may result in such a cost function (Sun and Tsai [1 Apart from direct applications, piecewise linear functions are often used to approximate functions th more difficult to handle, and therefore, solution procedures for $PLNP$ are of considerable interest.

$PLNP$ can be readily solved by a reformulation approach. In this approach each *piece* or cost se; for every arc in the network is replaced by an arc in an equivalent linear network flow problem (NP) resulting NP can be solved by any implementation of the bounded simplex method specialized for ne flow problems, but it is a much larger problem to solve. As the number of *pieces* in the $PLNP$ increa does the dimension of the NP resulting from the conversion.

The motivation of our study is to be able to efficiently solve piecewise linear approximations of no cost network flow problems. The approximation may have to be revised several times, and therefore, : it using a reformulation approach is undesirable. Such an approach is not only likely to take longer the increased dimensions but will also result in additional burden due to the data manipulation neede each revision.

The growth of dimensions can be avoided if a solution approach that can solve the *PLNP* dire its original form, is used. Fourer [1985,1988] and Premoli [1986] presented direct approaches for : piecewise linear programs. Fourer [1985,1988] gives a comprehensive discussion of such an approach. approaches can be specialized for problems with a network structure. Sun and Tsai [1990] impler Fourer's approach for network flow problems. The approach uses a bound on the *true* reduced cost nonbasic arcs. If none of the nonbasic arcs price favourably using the bound then an optimal solu guaranteed. The drawback is that when a nonbasic arc does price favourably, its *true* reduced cost be determined to ensure that its entry into the basis will result in an improvement of the objective fu This entails cycle traces in the network before each pivot can be made.

In this paper we outline a slightly different approach for solving the *PLNP* directly. Our approac not require the determination of the *true* reduced cost of a seemingly favourable nonbasic arc, and the cycle traces are not needed. From our empirical results, we conclude that this allows a significant red in the time taken to solve the *PLNP*. Section 2 gives an introduction to the problem followed by a de discussion of the *direct* method with our approach in Section 3. Our implementation is discussed in S 4 and the computational results are reported in Section 5. Piecewise linear modification of 30 benc problems given by Klingman et al. [1974] are used for the computational testing and their dimensio given in Table III. The problems are solved using both the reformulation and our direct approach. Sign reduction in computational times is achieved, as reported, when the latter approach is used.

2. Problem Definition

The problem of interest can be defined as follows:

$$\min \quad f(x) = \sum_{j=1}^{n} f_j(x_j)$$

$$\text{s.t.} \quad Ax = b \qquad (P.$$

$$0 \le x \le u$$

where $A \in \Re^{m \times n}$ is a given node-arc incidence matrix, $b \in \Re^m$ is the given requirement vector, $u \in$ the given upper bound vector, and

$$f_j(x_j) = \begin{cases} c_j^1, & \text{if } 0 \le x_j \le u_j^1; \\ \vdots & \\ c_j^{s_j}, & \text{if } u_j^{s_j-1} \le x_j \le u_j^{s_j}. \end{cases}$$

The arc e_j has s_j segments. The *breakpoints* $0, u_j^1, \ldots, u_j^{s_j}$ are such that $0 < u_j^1 < u_j^2 \ldots < u_j^{s_j} = u$ capacity of the arc e_j). The costs $c_j^1, \ldots, c_j^{s_j}$ on the s_j segments of arc e_j are increasing, i.e.

$$0 < c_j^1 < c_j^2 \ldots < c_j^{s_j} < \infty$$

In the reformulation approach the j^{th} arc would be replaced by a total of s_j arcs. We solve the pr *directly* and, therefore, retain its current dimension.

3. The Algorithm

The algorithm can be seen as an extension of the upper-bounded simplex method for network program problems. The upper-bounded simplex method allows the flow in a nonbasic arc to be on its lower or u bound. In our case each arc is allowed to have several *breakpoints* and the flow in a nonbasic arc is all to be at any one of these *breakpoints*.

Definition: A *basic* solution $[x^B | x^N]$ is a partitioning of x, where x^B are the basic variables which can any value within their lower and upper bounds, and x^N are the nonbasic variables, which must take o value of one of the *breakpoints*.

Every basic solution corresponds to a rooted spanning tree T_B in the network (see Kennington Helgason [1980]). The j^{th} arc e_j is considered to be on the i^{th} segment if the flow x_j in it is such $u_j^{i-1} < x_j \le u_j^i$. If the flow on an arc is equal to zero, the arc is considered to be on the first segment if and on segment zero otherwise. The effective cost of any basic arc is unambiguously determined as the of the segment it is on. Using these costs, the dual vector π can be obtained from $\pi B = c_B$, where B i basis matrix.

Definition: For any *path* P,

$$P = \{s_1, e_1, s_2, e_2, \ldots, s_n, e_n, s_{n+1}\},$$

linking the from-node s_1 to the to-node s_{n+1} in T_B, the *orientation sequence* $O(P)$ is given by

$$O_i(P) = \begin{cases} +1, & \text{if } e_k = (s_k, s_{k+1}); \\ -1, & \text{if } e_k = (s_{k+1}, s_k) \end{cases}$$

(see Kennington and Helgason [1980]).

3.1 *Pricing*

The *price* of a nonbasic arc is defined as

$$p^+_{N_j} = c^+_{N_j} - \pi_F + \pi_T$$

$$p^-_{N_j} = c^-_{N_j} - \pi_F + \pi_T$$

where π_F and π_T are the duals of the from-node and the to-node respectively, of the arc e_{N_j}. $c^+_{N_j}, c^-_{N}$ the slopes on the right and left, respectively, of the flow in the arc. The pricing operation can be se an extension of that in the upper-bounded simplex method. Unless a nonbasic variable x_{N_j} is at its or upper bound, it can enter the basis if either increasing or decreasing its value will improve the obje function, i.e., $p^+_{N_j} < 0$ and x_{N_j} is increased, or $p^-_{N_j} > 0$ and x_{N_j} is decreased. Thus, the i^{th} and the $(i$ - segments of each nonbasic arc are priced to determine if either of them prices favourably. In the case v the arc is at its lower (upper) bound, the flow in the arc cannot decrease (increase) any further, and there only the first (last) segment is priced.

Proposition 3.1 For a nonbasic arc on its i^{th} segment, it suffices to price only the i^{th} and the $(i +$ segments of each nonbasic arc. If these segments do not price favourably, the segments $1, 2, \ldots, (i-1)$ $2), \ldots, s_j$ for the nonbasic arc e_j will not price favourably.

Proof: The result clearly follows from the fact that the piecewise linear costs on each arc are conve

This pricing strategy can be viewed as a *restricted basis entry* approach as opposed to the pricing nonbasic arcs in the reformulation approach. Note that the price p_{N_j} gives the *true* reduced cost of th e_{N_j} only when none of the basic arcs are at *breakpoints*, or equivalently, the solution is nondegenerate.

The *true* reduced cost of a nonbasic arc e_{N_j} is given by:

$$d^+_{N_j} = c^+_{N_j} - \sum_{i \in B} (c_B)_i (B^{-1} A_{.N_j})_i$$

$$d^-_{N_j} = c^-_{N_j} - \sum_{i \in B} (c_B)_i (B^{-1} A_{.N_j})_i$$

where $d^+_{N_j}$ is the reduced cost when the flow in the arc is to be increased, and $d^+_{N_j}$ is the reduced co the flow is to be decreased.

<u>*Proposition 3.2*</u> When some of the basic arcs are at *breakpoints*, $d^+_{N_j} > p^+_{N_j}$ and $d^-_{N_j} < p^-_{N_j}$, nonbasic arc e_{N_j}.

<u>*Proof :*</u> Let $A_{.N_j} = Y_j$. Then

$$d^+_{N_j} = c^+_{N_j} + \sum_{(Y_j)_i = -1} (c^+_B)_i - \sum_{(Y_j)_i = 1} (c^-_B)_i$$

$$d^-_{N_j} = c^-_{N_j} + \sum_{(Y_j)_i = -1} (c^-_B)_i - \sum_{(Y_j)_i = 1} (c^+_B)_i .$$

If some of the basic arcs did not have a possibly underestimated cost assigned to them, p_{N_j} as defined would give the net change in cost per unit change of flow in the nonbasic arc being priced (see [3]). the assigned costs can only underestimate the actual costs, p_{N_j} and d_{N_j} will differ in costs of only basic arcs e_j which will have an increment in flow due to the incoming nonbasic arc. Let Ω denote the indices of all such arcs. This implies that for $i \in \Omega$,

$$d^+_{N_j} = p^+_{N_j} + \sum_{(Y_j)_i = -1} (c^+_B)_i - (c^-_B)_i$$

$$d^-_{N_j} = p^-_{N_j} - \sum_{(Y_j)_i = -1} (c^+_B)_i - (c^-_B)_i$$

Since the piecewise linear costs are convex, $(c^+_B)_i - (c^-_B)_i > 0$ for all i. Therefore,

$$d^+_{N_j} > p^+_{N_j}$$

$$d^-_{N_j} < p^-_{N_j}$$

If $p^+_{N_j} \geq 0$ and $p^-_{N_j} \leq 0$, the *true* reduced cost of the nonbasic arc e_{N_j} cannot be favourable as well. The if $p^+_{N_j} \geq 0$ and $p^-_{N_j} \leq 0$ for all nonbasic arcs, the solution is optimal.

3.2 *Ratio Test*

The amount by which the flow is allowed to change on any arc is the difference in the value of the n *breakpoint* and the flow. That is, the flow in a basic arc is allowed to move up or down only on the c segment. This differs from the strategy used in Fourer [1985,1988] and implemented by Sun and T [1990], where at a given iteration, flows can change over several segments. Our approach eliminates the to trace cycles on the tree since in case

$$p^+_{N_j} < 0, \quad \text{but} \quad d^+_{N_j} > 0$$

or

$$p^-_{N_j} > 0, \quad \text{but} \quad d^-_{N_j} < 0$$

we allow a pivot, which is degenerate.

Our computational experience demonstrates that the *direct* approach used by Sun and Tasi [19 ineffective since the cycle traces consume more time than that saved by keeping the dimension of the pro unchanged. The times taken by our implementation of the two approaches have been compared in Tal

3.3 *The Algorithm*

<u>STEP 0:</u>

Find an initial basic feasible solution $x^0 = [x^B|x^N]$ and let T_B be the basis tree.

Calculate π using $\pi B = c_B$. Initially $c_j = c_j^-(x_j^0)$ unless $x_j^0 = 0$, in which case $c_j = c_j^+$.

<u>STEP 1:</u> Pricing

Let

$$\Psi_1 = \{e_{N_j} : p^+_{N_j} < 0 \text{ and flow } x_{N_j} \text{ in arc } e_{N_j} \text{ is to be increased } \}$$

$$\Psi_2 = \{e_{N_j} : p^-_{N_j} > 0 \text{ and flow } x_{N_j} \text{ in arc } e_{N_j} \text{ is to be decreased.}\}$$

If $\Psi_1 \cup \Psi_2 = \phi$, terminate with $[x^B|x^N]$ as the optimal solution,

Otherwise select $e_j \in \Psi_1 \cup \Psi_2$ and set

$$\delta = \begin{cases} +1, & \text{if } e_j \in \Psi_1; \\ -1, & \text{if } e_j \in \Psi_2. \end{cases}$$

STEP 2: Ratio Test

Set

$$\Delta_1 = \begin{cases} u_j^+ - x_j, & \text{if } e_j \in \Psi_1; \\ x_j - u_j^-, & \text{if } e_j \in \Psi_2, \end{cases}$$

where $u_j^+ > x_j$ and $u_j^- < x_j$ are the closest *breakpoints* to the right and left of x_j, respectively.

$$\Delta_2 = \min_{O_i(P)=\delta} \{x_i - u_i^{k-1}\}$$

$$\Delta_3 = \min_{-O_i(P)=\delta} \{u_i^k - x_i\}$$

where

$$u_i^{k-1} \leq x_i \leq u_i^k$$

Define

$$\Delta = \min\{\Delta_1, \Delta_2, \Delta_3\}$$

Let the minimum be attained for arc e_l

STEP 3: Update Flows

Set $x_j = x_j + \Delta\delta$ and $x_i = x_i - \Delta\delta O_i(P)$

If $\Delta = \Delta_1$ go to step 1

STEP 4: Tree and Dual Update

Replace e_l in the tree by e_j. Update B, N and the duals.

Return to step 1.

4. Implementation

NETFLO [1980], which is an efficient implementation of the upper bounded simplex method on the g has been modified to implement the *direct* simplex algorithm and the modified code is called RFSBAS data structures are, therefore, an extension of those used in NETFLO. RESBAS has additional arc-l arrays used to store the *breakpoints* and costs for the s_j segments of each arc e_j. Another arc-length ar used to indicate the current segment that each arc is on. Additional node-length arrays are also intro to record the closest *breakpoints* above and below the current flow x_j for the basic arc e_j.

The strategy outlined by Fourer in [1988] and implemented by Sun and Tsai in [1990], where flo arc is allowed to change over more than one segment has also been implemented for a comparative The implementation is called PWNET.

5. Computational Experience

Standard network flow problems generated by NETGEN [1974] were modified to produce problem piecewise linear cost functions. The capacity of each arc is divided into n segments which are as increasing costs. The number of segments generated can be varied. All the repoited times are the a of the time taken over three runs. Table I lists the times taken to solve the modified NETGEN pie linear problems with eight segments for each arc. The problems were solved using NETFLO, RESBA PWNET. All the three codes are *integer* FORTRAN codes. To solve the problems using NETFLO of the eight segments on every arc is replaced by an arc to yield an equivalent problem with a linea function. The dimension of the reformulated problem is, therefore, eight times that of the original pie linear problem. The testing was done on the SEQUENT Symmetry S81 and the times reported ar clock times in seconds. The time taken for the input of the data and the output of the results is not inc.

Tafle I illustrates that on the average, the *direct* approach used in RESBAS takes 48 percent of th taken by the reformulation approach. A large part of the savings in time is accounted for by the fact th the average, RESBAS takes 34 percent of the total time in the pricing operation, while NETFLO tal percent of the total time. The *direct* approach used in PWNET was found to take more time than NE in most cases. On studying the time taken by different parts of PWNET we concluded that the cycle needed in this approach turn out to be expensive and result in its poorer performance.

Table II lists the comparative times taken by NETFLO, RESBAS, and PWNET to solve the NET problems with 80 percent of the arcs specified to be capacitated. The original NETGEN problems uncapacitated arcs in most cases. An uncapacitated arc has a large upper bound, and under the abse a capacity requirement, the *breakpoints* are large as well. As seen from Table II, having a large perce of capacitated arcs does not effect the performance of either code considerably. RESBAS takes 55 perc the time taken by NETFLO, on the average, for these problems and the performanoe of PWNET cont to be poorer than that of NETFLO in most cases. PWNET is not used for further testing.

Table III lists the comparative times taken by NETFLO and RESBAS to solve NETGEN problems

60 percent of the arcs having a high cost and 50 percent being capacitated. The original NETGEN p have only 30 percent of the arcs with high costs in less than half of the problems. The increased per of high cost arcs has nearly no effect on the performance of RESBAS. From Table III we see that R takes 52 percent of the time taken by NETFLO for these problems on the average.

A few problems were solved using NETFLO and RESBAS to see the effect of varying the nur segments used. The problems were solved using 4, 8, 16, 24, and 32 segments. The results are tabul Table IV, and figures 1-5 illustrate the comparative time taken by the two codes. As the number of se used is increased, the relative performance of RESBAS is seen to improve. This is expected since w increase in the number of segments the dimension of the problem to be solved by NETFLO becomes which results in RESBAS having an increasing advantage in the time needed to solve the problem. (average for the five problems, RESBAS takes approximately 71 percent of the time taken by NETFLC the number of segments used is 4. With an increase in the number of segments used to 32, the adv that RESBAS has over NETFLO increases significantly, and the average time taken is seen to be o percent of that taken by NETFLO.

6. Conclusions

From our computational experience with problems of varying size and characteristics, we conclude *direct* solution procedure for $PLNP$ has a definite advantage over a reformulation approach. On the a RESBAS requires only 50 percent of the time required by NETFLO to solve the problems tested. T for some problems the improvement was seen to be less striking than the average, RESBAS does no longer than NETFLO to solve any of the problems. Since most of the savings in the solution time from the pricing operation, RESBAS is likely to have an increasing advantage as the density of the pr being solved increases. The number of arcs needed in the reformulation approach to replace an arc original problem also plays an important role in the extent to which the *direct* approach out perforn reformulation appraoch. As seen from tables 4-8 and figures 1-5, the improvement in the time tak RESBAS becomes increasingly significant as the number of segments used on each arc is increased f to 32.

6. References

1 R. Fourer, *Mathematical Programming* , 33, 204-233 (1985).

2 R. Fourer. *Mathematical Programming* , 41, 281-315 (1988).

3 J.L. Kennington and R.V. Helgason, *Algorithms for Network Programming* (Wiley-Interscien York, 1980).

4 D. Klingman, A. Napier and J. Stutz, *Management Science*, 20, N0. 5, 814-821 (1974).

5 A. Premoli, *Mathematical Programming* , 36, 210-227 (1986).

6 R.T. Rockafellar, *Network Flows and Monotropic Optimization* (Wiley-Interscience, New York,

7 J. Sun and K.-H. Tsai, *Working Paper*, (Northwestern University, Evanston, IL, 1990).

Table I TIME‡ TAKEN FOR ORIGINAL PROBLEMS

PRB NUM	NETFLO	RESBAS	TIME RATIO*	PWNET	TIME RATIO*	OPTIMAL VALUE
1	2.17	.82	.38	2.02	.93	209934
2	2.28	.83	.37	1.98	.87	180820
3	3.77	1.31	.35	2.81	.75	162949
4	3.16	1.11	.35	2.39	.76	124369
5	4.58	1.46	.32	2.69	.59	130774
6	5.65	2.24	.40	5.60	.99	203737
7	9.27	3.16	.34	7.08	.76	161329
8	9.55	3.19	.33	6.91	.72	161288
9	11.35	3.77	.33	7.28	.64	141902
10	11.51	3.68	.32	7.63	.66	190070
11	2.92	1.49	.51	4.19	1.43	8871025
12	3.90	1.49	.38	3.23	.83	3682451
13	2.47	1.24	.50	3.52	1.43	7921016
14	3.51	1.39	.40	2.99	.85	3580569
15	4.25	2.52	.59	6.67	1.57	10660812
16	6.03	2.43	.40	5.14	.85	3145173
17	4.01	2.27	.57	4.62	1.15	10149301
18	4.58	1.95	.43	2.97	.65	2814656
19	7.02	4.18	.60	9.37	1.33	16522487
20	11.73	6.14	.52	14.03	1.20	19403203
21	5.11	2.78	.54	4.52	.89	11955635
22	8.20	3.77	.46	6.72	.82	14674811
23	6.10	3.13	.51	10.12	1.66	14471077
24	7.11	3.82	.54	10.84	1.52	12417227
25	9.17	4.42	.48	13 19	1.44	9154491
26	14.20	8.10	.57	25.17	1.77	16534913
27	14.59	7.88	.54	28.85	1.98	16363553
28	267.20	243.92	.91	1466.85	5.49	104874927
29	197.13	162.15	.82	791.95	4.02	37115244
30	125.51	71.32	.57	290.27	2.31	6462989
		Average Ratio	.48		1.36	

‡Wall clock seconds on the Sequent Symmetry S81 *With respect to NETFLO

Table II TIME[‡] TAKEN WITH 80% ARCS CAPACITATED

PRB NUM	NETFLO	RESBAS	TIME RATIO*	PWNET	TIME RATIO*	OPTIMAL VALUE
1	3.10	1.29	.42	3.46	1.12	234607
2	3.84	1.52	.40	3.44	.90	218543
3	5.06	1.87	.37	3.99	.79	203033
4	5.43	1.73	.32	3.90	.72	164195
5	7.76	2.54	.33	4.14	.53	108281
6	9.46	3.73	.39	9.04	.95	296534
7	12.38	4.46	.36	8.88	.72	204810
8	13.50	5.14	.38	8.63	.64	139952
9	15.85	5.53	.35	10.59	.67	131246
10	14.73	4.93	.33	8.55	.58	136127
11	5.35	3.37	.63	8.28	1.55	10889306
12	10.61	5.88	.55	9.17	.86	5186373
13	4.16	2.43	.58	4.95	1.19	9138062
14	6.15	2.84	.46	5.44	.88	4021478
15	7.06	4.55	.65	10.01	1.42	12538308
16	11.36	5.88	.52	10.93	.96	4259552
17	5.16	3.07	.59	6.13	1.19	8788133
18	6.98	3.25	.47	5.21	.75	3214784
19	7.02	4.18	.60	9.37	1.33	16522487
20	11.73	6.14	.52	14.03	1.20	19403203
21	5.11	2.78	.54	4.52	.89	11955635
22	8.20	3.77	.46	6.72	.82	14674811
23	20.82	15.35	.74	36.38	1.75	21816035
24	24.37	17.12	.70	40.92	1.68	18923270
25	31.21	23.47	.75	63.16	2.02	17110810
26	50.75	39.14	.77	97.90	1.93	25418455
27	52.81	44.33	.84	111.86	2.12	25106519
28	356.22	326.03	.92	1646.93	4.62	103360085
29	269.22	235.88	.88	1024.10	3.80	38642574
30	158.95	97.92	.62	356.02	2.24	7154302
		Average Ratio	.55		1.36	

[‡] Wall clock seconds on the Sequent Symmetry S81 *With respect to NETFLO

Table III TIME[‡] TAKEN WITH 60% HIGH COST ARCS

PRB NUM	# OF NODES	# OF ARCS	NETFLO	RESBAS	TIME RATIO	OPTIMAL VALUE
1	200	1300	3.04	1.28	.42	274313
2	200	1500	3.34	1.33	.40	184679
3	200	2000	4.28	1.67	.39	213952
4	200	2200	4.53	1.67	.37	162210
5	200	2900	5.61	1.90	.34	184942
6	300	3150	9.42	3.89	.41	348180
7	300	4500	12.42	4.46	.36	245771
8	300	5155	10.67	3.82	.36	176457
9	300	6075	16.18	5.27	.33	157974
10	300	6300	13.83	4.57	.33	199056
11	400	1306	3.80	2.18	.57	10513520
12	400	2443	5.44	2.42	.44	3787240
13	400	1306	3.59	2.04	.57	9568307
14	400	2443	4.68	2.02	.43	3534618
15	400	1416	4.26	2.45	.57	9032347
16	400	2836	7.77	3.40	.44	4211578
17	400	1416	3.89	2.05	.53	8445932
18	400	2836	6.33	2.69	.42	3744264
19	400	1382	7.72	4.84	.63	30869990
20	400	2676	11.93	6.22	.52	13107867
21	400	1382	5.30	2.96	.56	19048864
22	400	2676	8.13	3.68	.45	10090412
23	1000	2900	15.58	10.61	.68	27878120
24	1000	3400	13.99	9.39	.67	19301285
25	1000	4400	23.86	15.49	.65	16501324
26	1500	5107	34.69	24.92	.72	26560879
27	1500	5730	37.59	28.04	.75	25442763
28	8000	15000	317.98	294.26	.93	158984551
29	5000	23000	218.67	183.66	.84	46185509
30	3000	35000	135.21	82.97	.61	6713147
				Average Ratio	.52	

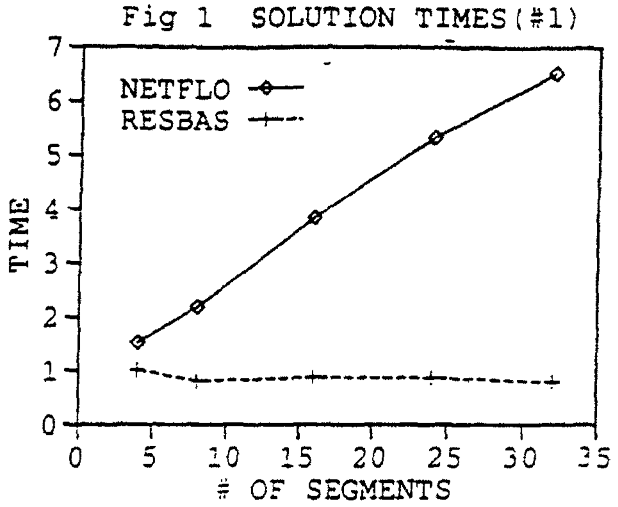
[‡] Wall clock seconds on the Sequent Symmetry S81

Table IV INCREASING # OF SEGMENTS

PRB	NUM	NETFLO	RESBAS	TIME
NUM	OF SEGS	TIME[‡]	TIME[‡]	RATIO
1	4	1.52	1.01	.67
	8	2.18	.81	.37
	16	3.84	.88	.23
	24	5.32	.87	.16
	32	6.52	.79	.12
10	4	6.50	3.75	.58
	8	11.51	3.69	.32
	16	19.66	3.44	.18
	24	25.33	2.76	.11
	32	25.67	2.19	.09
15	4	2.41	1.82	.76
	8	4.26	2.51	.59
	16	10.28	4.17	.41
	24	13.40	4.36	.33
	32	17.54	4.87	.28
20	4	5.30	3.80	.72
	8	11.77	6.12	.52
	16	30.14	10.55	.35
	24	60.18	17.12	.28
	32	94.71	24.32	.26
27	4	11.96	9.69	.81
	8	14.43	7.88	.55
	16	26.39	9.93	.38
	24	41.05	11.31	.28
	32	65.59	15.73	.24

[‡] Wall clock seconds on the Sequent Symmetry S81

Graphs of the Times Taken with Increasing Number of Segments



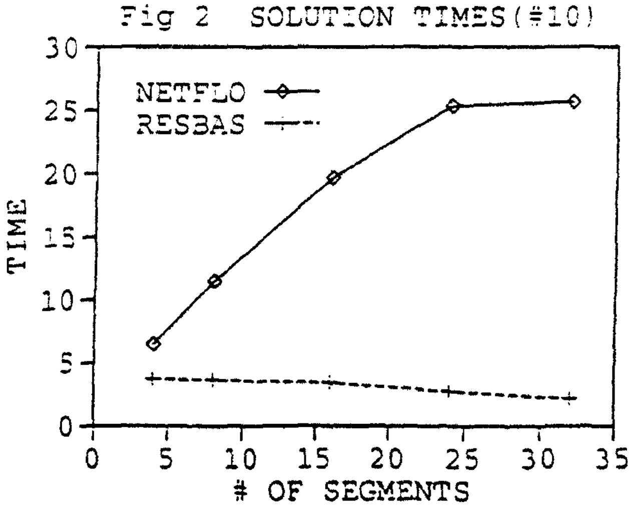


Fig 3 SOLUTION TIMES(#15)

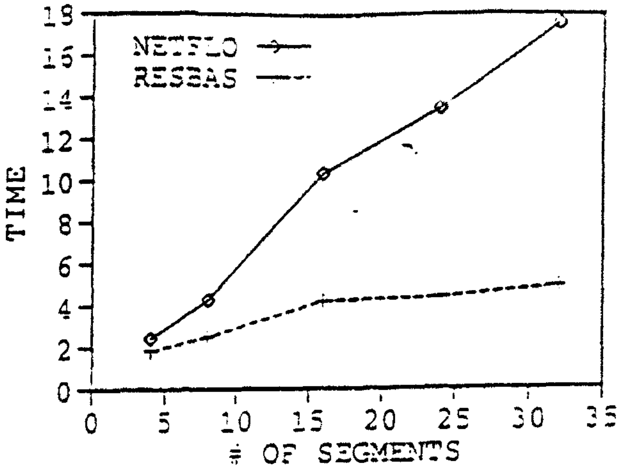


Fig 4 SOLUTION TIMES(#20)

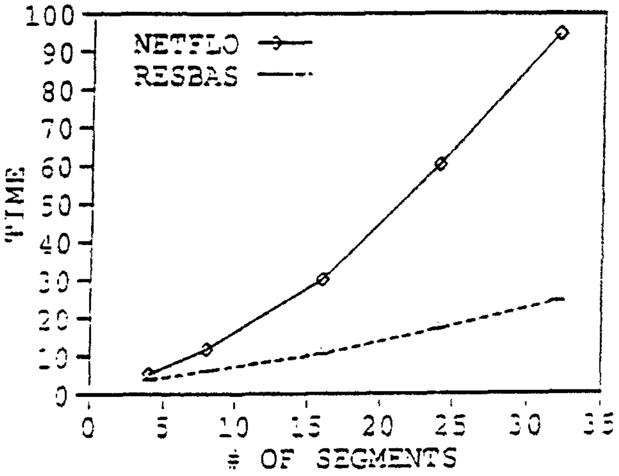
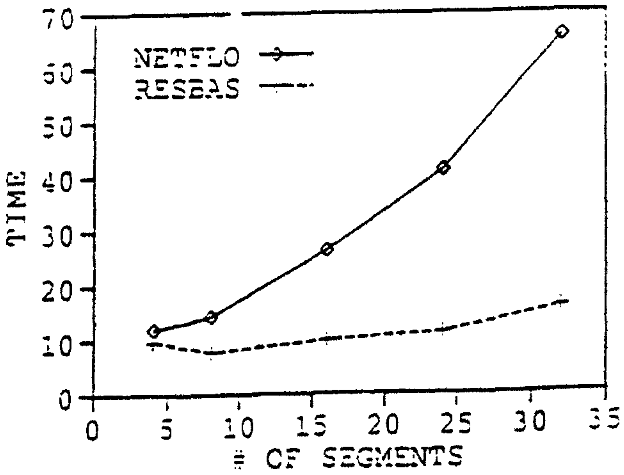


Fig 5 SOLUTION TIMES(#27)


NETFLO
RESBAS
TIME
# OF SEGMENTS